

**IN THE UNITED STATES DISTRICT COURT  
FOR THE NORTHERN DISTRICT OF ILLINOIS  
EASTERN DIVISION**

KOVE IO, INC.,

Plaintiff,

v.

GOOGLE LLC,

Defendant.

Civil Action No. 1:23-cv-04244

Jury Trial Demanded

**FIRST AMENDED COMPLAINT**

Plaintiff Kove IO, Inc. (“Kove”) files this First Amended Complaint against Defendant Google LLC (“Google”), and in support thereof states as follows:

**THE PARTIES**

1. Kove was founded with an emphasis on high performance storage solutions. Kove has maintained a lean, focused team over the course of its history, including individuals with strong backgrounds from premier tech companies, such as Intel, Sun, Cisco, Seagate, Quantum, NASA, and many others. It owns the patents asserted in this case.

2. Kove is a corporation organized and existing under the laws of the State of Delaware and is registered to do business in the State of Illinois. Kove’s principal place of business is located at 14 North Peoria Street Suite 2H, Chicago, Illinois 60607.

3. Google LLC is a wholly-owned subsidiary of Alphabet, Inc. On information and belief, Google is a Delaware limited liability company with a principal place of business at 1600 Amphitheatre Parkway, Mountain View, California 94043. Google LLC is registered to do business in the State of Illinois and has established places of business in this District, including at 320 N Morgan St, Unit 600, Chicago, IL 60607.

4. Google develops and provides cloud storage products and services to customers, including customers in this District.

### **JURISDICTION AND VENUE**

5. This Court has subject matter jurisdiction over this case under 28 U.S.C. §§ 1331 and 1338. Venue is proper in this Court under 28 U.S.C. §§ 1391 and 1400(b).

6. This Court has personal jurisdiction over Google because Google has committed acts in this District giving rise to this cause of action and has continuous and systematic business contacts within this District and the State of Illinois. Google creates products and services that are and have been used, offered for sale, sold, and purchased in the Northern District of Illinois, and Google has committed, and continues to commit, acts of infringement in the Northern District of Illinois, has conducted business in the Northern District of Illinois, and has engaged in continuous and systematic activities in the Northern District of Illinois.

7. Venue is proper in this District under 28 U.S.C. §§ 1391 and 1400(b). Google is registered to do business in Illinois. Additionally, upon information and belief, Google has transacted business in this District and has committed acts of direct infringement in this District by, among other things, making, using, offering to sell, selling, and importing products that infringed the Asserted Patents. Google has regular and established places of business in the District, including at 320 N Morgan St, Unit 600, Chicago, IL 60607. Further, this District has significant interest in this case, as Google has a long-standing presence in the Northern District of Illinois, and on information and belief, Google's employees in the Northern District of Illinois have relevant knowledge about the Accused Products in this case.

### **BACKGROUND**

8. Kove's inventors developed breakthrough technology enabling high-performance, hyper-scalable distributed storage years before the advent of the cloud. Kove was awarded patents

for its innovations, and in 2004, the MIT Technology Review ranked the distributed data storage technology described in Kove's patents as one of the top 10 emerging technologies that would change the world.<sup>1</sup>

9. Drs. Overton and Bailey, who met at the University of Chicago while working on their PhDs, are the named inventors of the three Asserted Patents: U.S. Patent Nos. 7,814,170 (the "'170 Patent"); 7,103,640 (the "'640 Patent"); and 7,233,978 (the "'978 Patent") (collectively, the "Asserted Patents").

10. When the inventors were studying together at the University of Chicago in the 1990's, they foresaw the need for storing large amounts of data in scalable, distributed storage networks – what is now called in common vernacular, "the cloud." The inventors identified a key roadblock that was needed for cloud storage to be the viable and ubiquitous technology that it is today.

11. In particular, data storage management has become increasingly critical for companies as the amount of data produced every day grows exponentially—for example, in 2017, an estimated 2.5 *quintillion* bytes of data were being created each day.<sup>2</sup> That amount has grown approximately 70 times larger, with estimated that 64.2 *zettabytes* (trillions of gigabytes) were created in all of 2020.<sup>3</sup> In the 1990's, the inventors foresaw that data storage requirements would

---

<sup>1</sup> Ex. 1, Balakrishnan, H., Distributed Storage, 10 Emerging Technologies That Will Change Your World, MIT Technology Review February 2004.

<sup>2</sup> Ex. 2, Bernard Marr, How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read (May 21, 2018), available at <https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/?sh=5868549760ba>; see also Ex. 3, *How Much is 2.5 Quintillion?*, MEDIUM (Feb. 8, 2017), available at <https://medium.com/@nicole.chardenet/how-much-is-2-5-quintillion-361aff053059> (noting that "2.5 quintillion pennies would, if laid flat, cover the Earth five times").

<sup>3</sup>Ex. 4, *Big Data for Sustainable Development*, United Nations Global Issues report, <https://www.un.org/en/global-issues/big-data-for-sustainable-development> (estimating that 64.2

grow beyond the capabilities of conventional computer networks. The inventors identified a key issue: the dependency on the conventional ways of indexing the information in a distributed storage network. For example, as information was added to a network, previous technology would require that it be indexed in some hierarchical fashion, and those indices would periodically have to be updated. While these indices worked well to a certain point, the distributed network of data servers and data would become so vast and complex that ordinary hierarchical indices no longer could handle the load without high levels of inefficiency.

12. Further, to store a piece of information (e.g., a data file), a storage system must store not only the data file itself but also its corresponding location information, which records where the data file is located on the network of servers and computers. Without the corresponding location information, a storage system would not know where to find the data file (i.e., which server on the network to access and from which to retrieve the data file). At the time of the Drs. Overton and Bailey's inventions, location information of data files was conventionally stored on a single centralized server, and it was retrieved from that specific centralized server. Then-existing systems allowed a client seeking location information associated with a data file to send a request to a server, and typically, only data statically associated with that server was returned. The search was conducted only where the system knew in advance to look.

13. In the 1990's, such a server may have been sufficient; however, the inventors knew distributed storage systems would someday contain so many unique data files that it would become impractical—if not impossible—to store the corresponding location information in one place.

---

zettabytes were created in 2020, which averages to approximately 175 exabytes or quintillions of bytes per day)

14. Drs. Overton and Bailey addressed these challenges. They realized, among other things, that storing location information associated with data files across multiple servers would reduce the processing time to find a data file. Likewise, the inventors understood the need to efficiently identify which of the multiple location information servers stored the location information for a particular data file. The inventors further understood the need to balance location information among distributed location servers to ensure that the distributed location solutions could be scalable and keep up with rapid growth of stored data.

15. To achieve a scalable, distributed location information system capable of handling massive growth in data storage needs, Drs. Overton and Bailey invented the claimed technology of the Asserted Patents. The claimed inventions were ahead of their time—distributed storage networks had not yet grown large enough. But today, distributed storage systems are ubiquitous, and the inventions of Drs. Overton and Bailey undergird their capacity and viability.

#### **THE KOVE ASSERTED PATENTS**

16. On October 12, 2010, the U.S. Patent and Trademark Office duly and legally issued the '170 Patent, entitled "Network Distributed Tracking Wire Transfer Protocol," with Drs. Overton and Bailey as inventors. A true and correct copy of the '170 Patent is attached as Exhibit 5.

17. On June 19, 2007, the U.S. Patent and Trademark Office duly and legally issued the '978 Patent, entitled "Method and Apparatus for Managing Location Information in a Network Separate From the Data to Which the Location Information Pertains," with Drs. Overton and Bailey as inventors. A true and correct copy of the '978 Patent is attached as Exhibit 6.

18. On September 5, 2006, the U.S. Patent and Trademark Office duly and legally issued the '640 Patent, entitled "Network Distributed Tracking Wire Transfer Protocol," with Drs.

Overton and Bailey as inventors. A true and correct copy of the '640 Patent is attached as Exhibit 7.

19. Kove is the sole and exclusive owner of all rights, title, and interest to the Asserted Patents as necessary to bring this action, including the right to recover past and future damages.

20. The Asserted Patents are enforceable and valid.

### **GOOGLE'S INFRINGING PRODUCTS AND SERVICES**

21. Upon information and belief, Google has directly infringed one or more claims of the Asserted Patents.

22. The accused products include without limitation cloud-based products and services provided by Google, such as Google Spanner and Google Colossus (collectively, the "Google Accused Products" or the "Accused Products").

23. As an example, Google Spanner is a "globally-consistent, scalable relational database."<sup>4</sup> Google Spanner helps "provide the underlying infrastructure for all Google Cloud storage services" by "stor[ing] all the metadata about access permissions and data location."<sup>5</sup> Google touts Spanner as "the only enterprise-grade, globally-distributed, and strongly-consistent database service built for the cloud."<sup>6</sup> As such, Spanner "provide[s] high availability and scalability" and "global consistency"<sup>7</sup> to "hundreds of mission-critical services at Google."<sup>8</sup>

---

<sup>4</sup> Ex. 8. *See Colossus Under the Hood: A Peek Into Google's Scalable Storage System*, GOOGLE (Apr. 19, 2021), available at <https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system>.

<sup>5</sup> *Id.*

<sup>6</sup> Ex. 9. *What is Cloud Spanner?*, GOOGLE (June 1, 2021), available at <https://cloud.google.com/blog/topics/developers-practitioners/what-cloud-spanner>.

<sup>7</sup> *Id.*

<sup>8</sup> Ex. 10. Bacon, David F., et al., *Spanner: Becoming a SQL System*, at 1 GOOGLE, INC. (2017), available at <https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/46103.pdf>.

24. Spanner stores data in Google Colossus, another Accused Product that helps provides critical cloud services.<sup>9</sup> Google Colossus is a distributed file storage system that provides the underlying infrastructure permitting myriad systems to store data in the cloud.<sup>10</sup> Colossus “helps enable massive scalability and data durability for Google services as well as . . . applications.”<sup>11</sup> Google considers Colossus one of the “three main building blocks used by each of [Google’s] storage services,” as it “manages, stores, and provides access to all [user] data.”<sup>12</sup> Colossus manages the hardware running in Google’s data centers, helps move information between data storage and applications that use the data, and helps provide high reliability for the Google Cloud storage products and data centers.<sup>13</sup>

25. Google Colossus enables massive scalability, cheaper storage options, and optimal storage efficiency.<sup>14</sup> Colossus provides reliable and redundant storage,<sup>15</sup> and processing in Colossus “happens almost instantly.”<sup>16</sup> “Colossus underpins Google Cloud Storage,”<sup>17</sup> and “Google Cloud . . . underpins Google’s most popular products, supporting globally available

---

<sup>9</sup> *Id.*; see also *Under the Hood of Google Cloud Data Technologies: BigQuery and Cloud Spanner*, GOOGLE CLOUD TECH (Nov. 5, 2021), at 2:15, available at <https://www.youtube.com/watch?v=HhXkCjROvGE>.

<sup>10</sup> Ex. 8; see *Colossus Under the Hood: A Peek Into Google’s Scalable Storage System*, GOOGLE (Apr. 19, 2021), available at <https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system>.

<sup>11</sup> *Id.*

<sup>12</sup> *Id.*

<sup>13</sup> Ex. 11. Maria Deutscher, *Google Shares Technical Overview of its Exabyte-Scale Colossus File System*, SILICON ANGLE (Apr. 19, 2021), available at <https://siliconangle.com/2021/04/19/google-shares-technical-overview-exabyte-scale-colossus-file-system/>.

<sup>14</sup> Ex. 12. Denis Serenyi, *Cluster-Level Storage @ Google*, GOOGLE (Nov. 13, 2017) at 16, 27, available at <http://www.pdsw.org/pdsw-discs17/slides/PDSW-DISCS-Google-Keynote.pdf>.

<sup>15</sup> *Id.*

<sup>16</sup> Ex. 13. Cade Metz, *Google Remakes Online Empire with ‘Colossus,’* WIRED (July 10, 2012), available at <https://www.wired.com/2012/07/google-colossus/>.

<sup>17</sup> *Id.*

services like YouTube, Drive, and Gmail.”<sup>18</sup> As such, Google Colossus helps power “ongoing revenue growth from Google Cloud” and an empire of nearly \$258 billion of annual revenue.<sup>19</sup>

**COUNT I**  
**INFRINGEMENT OF THE '170 PATENT**

26. Kove incorporates by reference the foregoing paragraphs of this Complaint as if fully set forth herein.

27. On information and belief, Google has infringed the '170 Patent. Google directly infringed the '170 Patent under 35 U.S.C. §271(a) by making, using, selling, offering for sale, and/or importing in this District and into the United States products and/or methods covered by one or more claims of the '170 Patent, including, but not limited to, the Google Accused Products.

**Google Spanner Infringed the '170 Patent.**

28. As an example, Google Spanner infringed at least claim 1 of the '170 Patent.

29. Claim 1 of the '170 Patent is directed to a “system for managing data stored in a distributed network.” Claim 1, for example, recites “a data repository configured to store a data identity” and “a data location server network . . . wherein data location information for a plurality of data entities is stored in the data location server network.”

30. Claim 1, in its entirety, recites:

1. A system for managing data stored in a distributed network, the system comprising:
  - a data repository configured to store a data entity, wherein an identifier string identifies the data entity; and
  - a data location server network comprising a plurality of data location servers, wherein data location information for a plurality of data entities is stored in the data location server network, at least one of the plurality of data location servers

---

<sup>18</sup> Ex. 8. *Colossus Under the Hood*.

<sup>19</sup> Ex. 14. *See Alphabet Announces Fourth Quarter and Fiscal Year 2021 Results* (Feb. 1, 2022), available at

[https://abc.xyz/investor/static/pdf/2021Q4\\_alphabet\\_earnings\\_release.pdf?cache=d72fc76](https://abc.xyz/investor/static/pdf/2021Q4_alphabet_earnings_release.pdf?cache=d72fc76).



includes location information associated with the identifier string, each one of the plurality of data location servers comprises a processor and a portion of the data location information, the portion of the data location information included in a corresponding one of the data location servers is based on a hash function used to organize the data location information across the plurality of data location servers, and each one of the data location servers is configured to determine the at least one of the plurality of data location servers based on the hash function applied to the identifier string.

Ex. 5, Pg. 26-27.

31. Google Spanner provides a system for managing data stored in a distributed network. For example, Google Spanner is a globally distributed database system.

Databases are part of virtually every application you run in your organization and great apps need great databases. This post is focused on one such great database—Cloud Spanner.

Cloud Spanner is the only enterprise-grade, globally-distributed, and strongly-consistent database service built for the cloud, specifically to combine the benefits of relational database structure with non-relational horizontal scale. It is a unique database that combines transactions, SQL queries, and relational structure with the scalability that you typically associate with non-relational or NoSQL databases.

**Source:** Ex. 9, *What is Cloud Spanner?*, GOOGLE (June 1, 2021), available at <https://cloud.google.com/blog/topics/developers-practitioners/what-cloud-spanner>.

32. On information and belief, Google Spanner includes a data repository configured to store a data entity, wherein an identifier string identifies the data entity. For example, as illustrated in the screenshots below, Google Spanner provides a system for scalable, globally distributed databases. Google Spanner shares data across many sets of Paxos state machines in data centers spread all over the United States. Google Spanner divides user data, or data entities, into chunks called splits, and these splits are stored at different Cloud Spanner data repository servers present at different locations based on key ranges.

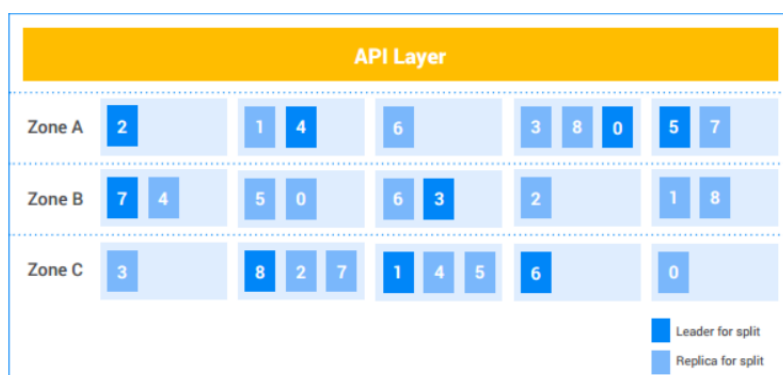
In this example, we have a table with a simple integer primary key.

Split	KeyRange
0	$[-\infty, 3)$
1	$[3, 224)$
2	$[224, 712)$
3	$[712, 717)$
4	$[717, 1265)$
5	$[1265, 1724)$
6	$[1724, 1997)$
7	$[1997, 2456)$
8	$[2456, \infty)$

Given the schema for `ExampleTable` above, the primary key space is partitioned into splits. For example: If there is a row in `ExampleTable` with an `Id` of `3700`, it will live in Split 8. As detailed above, Split 8 itself is replicated across multiple machines.

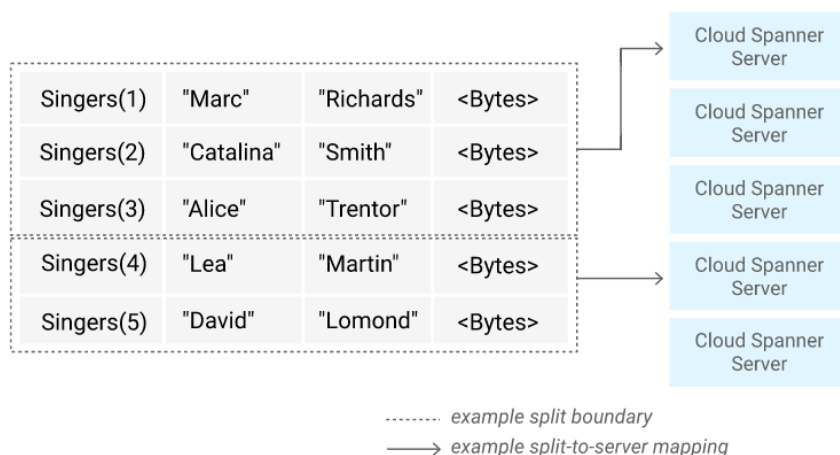
**Source:** Ex. 15, <https://cloud.google.com/spanner/docs/whitepapers/life-of-reads-and-writes>,

Page 4.



In this example Spanner instance, the customer has five nodes, and the instance is replicated across three zones. The nine splits are numbered 0-8, with Paxos leaders for each split being darkly shaded. The splits also have replicas in each zone (lightly shaded). The distribution of splits among the nodes may be different in each zone, and the Paxos leaders do not all reside in the same zone. This flexibility helps Spanner to be more robust to certain kinds of load profiles and failure modes.

**Source:** *Id.*, page 5.



**Source:** Ex. 16, <https://cloud.google.com/spanner/docs/schema-and-data-model>, Page 6.

33. Further, identifier strings identify the user data entities. For example, as illustrated in the screenshots below, user data in the form of a table must include a primary key. Google Spanner stores rows in sorted order by primary key values. Google's public product documentation teaches customers to choose primary keys that avoid hotspots and recommends techniques that can spread the load across multiple servers and avoid hotspots such as primary keys that are a combination of hashed ID and ID, serving as identifier strings used to identify the data entities stores in the Spanner database.

- [Hash the key](#) and store it in a column. Use the hash column (or the hash column and the unique key columns together) as the primary key.

**Source:** Ex. 16, <https://cloud.google.com/spanner/docs/schema-and-data-model>, Page 3.

In the table using the sequence, the key cannot simply be the numeric sequence value itself, as that will cause the last split to become a hotspot (as explained previously). So the application must generate a complex key that randomly distributes the rows among the splits.

This is known as *application-level sharding* and is achieved by prefixing the sequential ID with an additional column containing a value that's evenly distributed among the key space—e.g., a hash of the original ID, or bit-reversing the ID. That looks something like this:

```
01 CREATE TABLE Table1 (
02     Hashed_Id INT64 NOT NULL,
03     ID INT64 NOT NULL,
04     -- other columns with data values follow....
05 ) PRIMARY KEY (Hashed_Id, Id)
```

**Source:** Ex.17, <https://cloud.google.com/blog/products/gcp/what-dbas-need-to-know-about-cloud-spanner-part-1-keys-and-indexes>, Page 3.

34. On information and belief, Google Spanner includes a data location server network comprising a plurality of data location servers, wherein data location information for a plurality of data entities is stored in the data location server network, at least one of the plurality of data location servers includes location information associated with the identifier string, each one of the plurality of data location servers comprises a processor and a portion of the data location information, the portion of the data location information included in a corresponding one of the data location servers is based on a hash function used to organize the data location information across the plurality of data location servers, and each one of the data location servers is configured to determine the at least one of the plurality of data location servers based on the hash function applied to the identifier string. For example, as illustrated in the screenshots below, Google Spanner is a distributed database. It replicates the user data across multiple zones for availability and scale. Google Spanner divides data into chunks called splits, which are assigned to different

data location servers which can be at a different location and each zone holds a complete replica of the data.

database grows, Cloud Spanner divides your data into chunks called "splits", where individual splits can move independently from each other and get assigned to different servers, which can be in different physical locations. A split holds a range of contiguous rows. The start and end keys of this range are called "split boundaries". Cloud Spanner

**Source:** Ex. 16, <https://cloud.google.com/spanner/docs/schema-and-data-model>, Page 3.

unit of physical isolation: there may be one or more zones in a datacenter, for example, if different applications' data must be partitioned across different sets of servers in the same datacenter.

**Source:** Ex. 18, <https://storage.googleapis.com/pub-tools-public-publication-data/pdf/65b514eda12d025585183a641b5a9e096a3c4be5.pdf>, Page 3.

As a globally distributed database, Spanner provides several interesting features. First, the replication configurations for data can be dynamically controlled at a fine grain by applications. Applications can specify constraints to control which datacenters contain which data, how far data is from its users (to control read latency), how far replicas are from each other (to control write latency), and how many replicas are maintained (to control durability, availability, and read performance). Data can also be dynamically and transparently moved between datacenters by the system to balance resource usage across datacenters. Second, Spanner has two features that are difficult

**Source:** Ex. 18, <https://storage.googleapis.com/pub-tools-public-publication-data/pdf/65b514eda12d025585183a641b5a9e096a3c4be5.pdf>, Page 2.

35. Data location information for a plurality of data entities is stored in a network of a plurality of data location servers, and at least one of the plurality of data location servers includes location information associated with the identifier string. For example, the diagram excerpted from Google's published materials produced below, "illustrates the servers in a Spanner universe. A zone has one *zonemaster* and between one hundred and several thousand *spanservers*. The former assigns data to spanservers; the latter serve data to clients. The per-zone *location proxies* are used by clients to locate the spanservers assigned to serve their data." (Ex. 18 at 3.) For example, the diagram indicates that there are a plurality of location proxies per zone. By way of further example, there are a plurality (hundreds to thousands) of spanservers per zone.

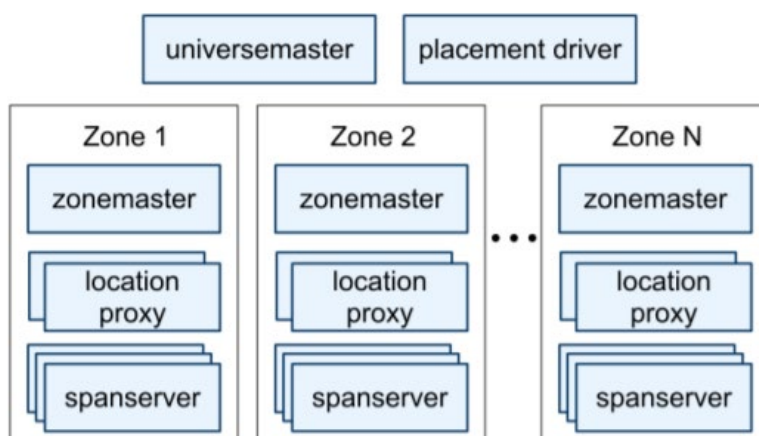


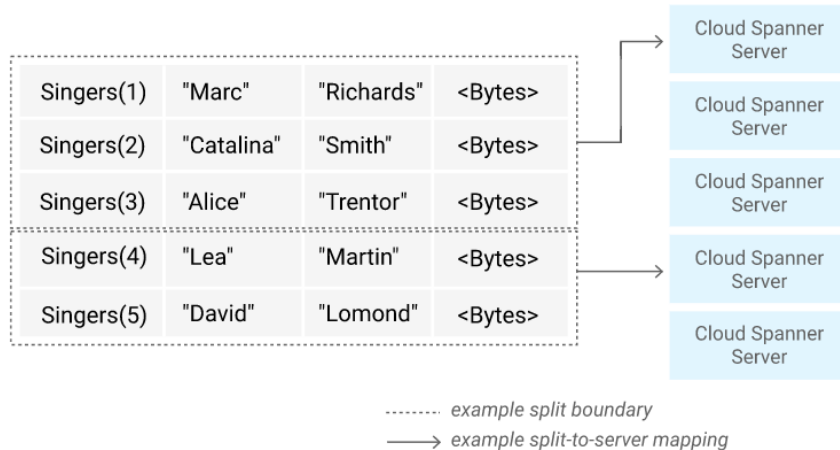
Fig. 1. Spanner server organization.

**Source:** Ex. 18, <https://storage.googleapis.com/pub-tools-public-publication-data/pdf/65b514eda12d025585183a641b5a9e096a3c4be5.pdf>, Page 3.

Figure 1 illustrates the servers in a Spanner universe. A zone has one *zonemaster* and between one hundred and several thousand *spanservers*. The former assigns data to spanservers; the latter serve data to clients. The per-zone *location proxies* are used by clients to locate the spanservers assigned to serve their data. The *universe master* and the *placement driver* are currently singletons. The universe master is primarily a console that displays status information about all the zones for interactive debugging. The placement driver handles automated movement of data across zones on the timescale of minutes. The placement driver periodically communicates with the spanservers to find data that needs to be moved, either to meet updated replication constraints or to balance load. For space reasons, we will only describe the spanserver in any detail.

**Source:** Ex. 18, <https://storage.googleapis.com/pub-tools-public-publication-data/pdf/65b514eda12d025585183a641b5a9e096a3c4be5.pdf>, Page 3.

36. By way of further example, as shown in the screenshots below, Google Spanner databases can contain one or more tables. One or more columns of the table are selected as primary keys, which uniquely identify each row in the table. Spanner partitions tables into contiguous key ranges called splits and divide data among servers by key ranges. The tables are split on boundaries between the rows, with the data from the resulting splits assigned to different Spanner servers based on key ranges, such as `hashed_id` and `id`.



**Source:** Ex. 16, <https://cloud.google.com/spanner/docs/schema-and-data-model>, Page 6.

undesirable because Cloud Spanner divides data among servers by key ranges, which means your inserts will be directed at a single server, creating a hotspot. There are techniques that can spread the load across multiple servers and avoid hotspots:

- [Hash the key](#) and store it in a column. Use the hash column (or the hash column and the unique key columns together) as the primary key.

**Source:** Ex. 16, <https://cloud.google.com/spanner/docs/schema-and-data-model>, Page 3.

There are some datasets that are too large to fit on a single machine. There are also scenarios where the dataset is small, but the workload is too heavy for one machine to handle. This means that we need to find a way of splitting our data into separate pieces that can be stored on multiple machines. Our approach is to partition database tables into contiguous key ranges called splits. A single machine can serve multiple splits, and there is a fast lookup service for determining the machine(s) that serve a given key range. The details of how data is split and what machine(s) it resides on are transparent to Spanner users. The result is a system that can provide low latencies for both reads and writes, even

**Source:** Ex. 15, <https://cloud.google.com/spanner/docs/whitepapers/life-of-reads-and-writes>,

Page 1.

37. On information and belief, each of the Google Spanner data location servers comprises a processor and a portion of the data location information, the portion of the data location information included in a corresponding one of the data location servers is based on a hash function used to organize the data location information across the plurality of data location servers. For example, as illustrated in the screenshots below, Google Spanner uses complex keys to randomly distribute, for example, hash, the rows of the aforementioned tables to avoid the



problem of “poorly distributed keys can result in hotspots, where a single node is responsible for the majority of reads and writes to the table.” (Ex. 15.)

In the table using the sequence, the key cannot simply be the numeric sequence value itself, as that will cause the last split to become a hotspot (as explained previously). So the application must generate a complex key that randomly distributes the rows among the splits.

This is known as *application-level sharding* and is achieved by prefixing the sequential ID with an additional column containing a value that’s evenly distributed among the key space—e.g., a hash of the original ID, or bit-reversing the ID. That looks something like this:

```
01 CREATE TABLE Table1 (
02     Hashed_Id INT64 NOT NULL,
03     ID INT64 NOT NULL,
04     -- other columns with data values follow....
05 ) PRIMARY KEY (Hashed_Id, Id)
```

**Source:** Ex. 17, <https://cloud.google.com/blog/products/gcp/what-dbas-need-to-know-about-cloud-spanner-part-1-keys-and-indexes>, Page 3.

is up-to-date enough; but a read that can be satisfied with stale data does not pay this latency. Another example is our data layout, where we use range sharding. Applications that do range scans can use this layout to get high performance. But some applications don't need range scans; and if those applications don't want to worry about the performance cost of potential row-range hotspots, they can put a hash value in their primary key, effectively causing Spanner to use hash sharding. In general, Spanner has been designed to offer powerful tools to application builders, but to give those builders a high degree of control over any tradeoffs between powerful functionality and performance.

**Source:** Ex. 15, <https://cloud.google.com/spanner/docs/whitepapers/life-of-reads-and-writes>,

Page 10.

38. On information and belief, each one of the data location servers is configured to determine the at least one of the plurality of data location servers based on the hash function applied to the identifier string. By way of further example, as illustrated in the below screenshots,



the hashed ID used as a part of the primary key is used by the fast lookup service as discussed above to locate the split that owns the key stored in one of the plurality of data location servers.

undesirable because Cloud Spanner divides data among servers by key ranges, which means your inserts will be directed at a single server, creating a hotspot. There are techniques that can spread the load across multiple servers and avoid hotspots:

- [Hash the key](#) and store it in a column. Use the hash column (or the hash column and the unique key columns together) as the primary key.

**Source:** Ex. 16, <https://cloud.google.com/spanner/docs/schema-and-data-model>, Page 3.

```
01  SELECT * FROM Table1
02  WHERE t1.Hashed_Id = 0xDEADBEEF
03          AND t1.Id = 1234
```

**Source:** Ex. 17, <https://cloud.google.com/blog/products/gcp/what-dbas-need-to-know-about-cloud-spanner-part-1-keys-and-indexes>, Page 3.

### **Google Colossus infringed the '170 Patent.**

39. As another example, Google Colossus infringed at least claim 15 of the '170 Patent.

40. Claim 15 of the '170 Patent is directed to a “method of handling location queries in a network.” Claim 15, for example, recites “a plurality of location servers including data location information,” “a plurality of locations in the network,” and a “plurality of identifiers identifying a respective one of a plurality of data entities.” Claim 15 further recites a “receiving a location query . . . requesting location information.” Further still, claim 15 recites “determining which of the plurality of location servers includes the location information;” “sending a location response message . . . comprising the location information;” and “sending a redirect message . . . in response to determining the one of the plurality of location servers fails to include the location information.”

41. Claim 15, in its entirety, recites:

15. A method of handling location queries in a network, the network comprising a plurality of location servers including data location information, the method comprising:

correlating each one of a plurality of identifiers with at least one of a plurality of locations in the network, each one of the plurality of identifiers identifying a respective one of a plurality of data entities, wherein the data entities are stored in corresponding locations in the network;

receiving a location query from a client at one of the plurality of location servers, the location query requesting location information identifying a location of a data entity included in the data entities;

determining which of the plurality of location servers includes the location information;

sending a location response message to the client in response to determining the one of the plurality of location servers includes the location information, the location response message comprising the location information; and

sending a redirect message to the client in response to determining the one of the plurality of location servers fails to include the location information, the redirect message identifying which of the plurality of location servers includes the location information.

42. Google Colossus provides a method of handling location queries in a network, the network comprising a plurality of location servers including data location information. For example, as shown below, Google Colossus is a distributed metadata system.

## Colossus in a nutshell

Now, let's take a closer look at how Colossus works.

But first, a little background on Colossus:

- It's the next-generation of the GFS.
- Its design enhances storage scalability and improves availability to handle the massive growth in data needs of an ever-growing number of applications.
- Colossus introduced a distributed metadata model that delivered a more scalable and highly available metadata subsystem.

**Source:** *Colossus Under the Hood: A Peek Into Google's Scalable Storage System*, GOOGLE (Apr. 19, 2021), available at <https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system>. Ex. 8.

43. On information and belief, Google Colossus correlates each one of a plurality of identifiers with at least one of a plurality of locations in the network, each one of the plurality of identifiers identifying a respective one of a plurality of data entities, wherein the data entities are stored in corresponding locations in the network. As explained above, Google Colossus stores metadata in Google BigTable, in which each of a plurality of identifiers identifies a respective one or a plurality of data entities. For example, as shown in the screenshots below, BigTable stores its data in an entity called an SSTable. An SSTable consists of a map from keys (i.e., a plurality of identifiers) to values (i.e., a plurality of data entities). A lookup operation in the SSTable uses a particular key to iterate over all of the key/value pairs in the SSTable to find the associated value.

keys and values are arbitrary byte strings. Operations are provided to look up the value associated with a specified

**Source:** Ex. 19, <https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

key, and to iterate over all key/value pairs in a specified key range. Internally, each SSTable contains a sequence

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

44. Further, Google Colossus correlates each one of a plurality of identifiers with at least one of a plurality of locations in the network, wherein the data entities are stored in corresponding locations in the network. For example, as illustrated in the screenshots below, each SSTable contains a sequence of blocks. An SSTable consists of a map from keys to values and at the end of it a block index is stored which is used to locate the block. BigTable performs a lookup with a single disk seek by first finding the appropriate block and then reading the appropriate block from the disk.

key range. Internally, each SSTable contains a sequence of blocks (typically each block is 64KB in size, but this is configurable). A block index (stored at the end of the SSTable) is used to locate blocks; the index is loaded into memory when the SSTable is opened. A lookup can be performed with a single disk seek: we first find the appropriate block by performing a binary search in the in-memory index, and then reading the appropriate block from disk. Optionally, an SSTable can be completely mapped into memory, which allows us to perform lookups and scans without touching disk.

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

45. On information and belief, Google Colossus receives a location query from a client at one of the plurality of location servers, the location query requesting location information

identifying a location of a data entity included in the data entities. For example, BigTable tablet server, or node acts as location server. As illustrated in the exemplary screenshots below, a request to recover a tablet is received at the tablet server, which accesses metadata in a Metadata that stores the location information of the tablet. The metadata contains, among other things, the list of SSTables that comprise a tablet. At the end of the SSTable a block index is stored, which identifies a location of a value and is used to locate the block when a location query is received.

### 5.1 Tablet Location

We use a three-level hierarchy analogous to that of a  $B^+$ -tree [10] to store tablet location information (Figure 4).

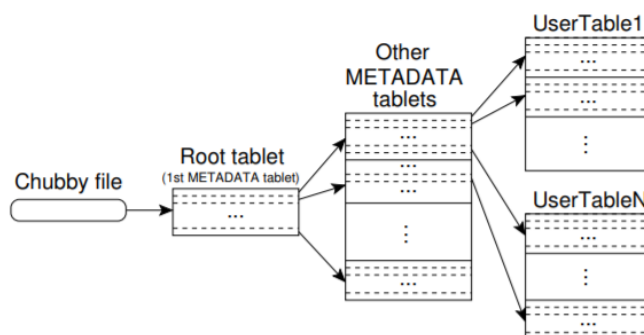


Figure 4: Tablet location hierarchy.

The first level is a file stored in Chubby that contains the location of the *root tablet*. The *root tablet* contains the location of all tablets in a special METADATA table. Each METADATA tablet contains the location of a set of user tablets. The *root tablet* is just the first tablet in the METADATA table, but is treated specially—it is never split—to ensure that the tablet location hierarchy has no more than three levels.

Source: Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en/archive/bigtable-osdi06.pdf>,

Page 4.

sequence of SSTables. To recover a tablet, a tablet server

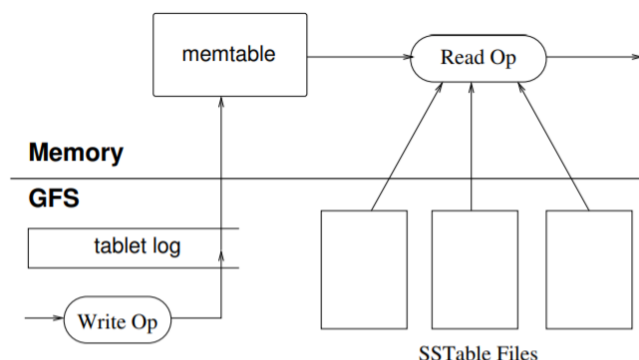


Figure 5: Tablet Representation

reads its metadata from the METADATA table. This meta-data contains the list of SSTables that comprise a tablet and a set of redo points, which are pointers into any commit logs that may contain data for the tablet. The

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 6.

key range. Internally, each SSTable contains a sequence of blocks (typically each block is 64KB in size, but this is configurable). A block index (stored at the end of the SSTable) is used to locate blocks; the index is loaded into memory when the SSTable is opened. A lookup can be performed with a single disk seek: we first find the appropriate block by performing a binary search in the in-memory index, and then reading the appropriate block from disk. Optionally, an SSTable can be com-

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

46. On information and belief, Google Colossus determines which of the plurality of location servers includes the location information. For example, as illustrated in the screenshots below, Google Colossus uses the selection made by a user to determine the route of the user's

request. The BigTable application profile (“app profile”) determines the route of the request between single-cluster routing and multi-cluster routing. The cluster is connected to one or more nodes, which are used to keep the record of the location of the tablet. *See also* <https://cloud.google.com/bigtable/docs/overview>, Ex. 20, Page 4 (“Each node in the cluster handles a subset of the requests to the cluster. By adding nodes to a cluster, you can increase the number of simultaneous requests that the cluster can handle. Adding nodes also increases the maximum throughput for the cluster. If you enable replication by adding additional clusters, you can also send different types of traffic to different clusters. Then if one cluster becomes unavailable, you can fail over to another cluster. . . . Each tablet is associated with a specific Bigtable node. . . . Importantly, data is never stored in Bigtable nodes themselves; each node has pointers to a set of tablets that are stored on Colossus.”).

### Settings in app profiles

An app profile defines the [routing policy](#) that Bigtable uses. It also controls whether [single-row transactions](#) are allowed.

#### Routing policy

An app profile specifies the routing policy that Bigtable should use for each request.

- **Single-cluster routing** routes all requests to 1 cluster in your instance. If that cluster becomes unavailable, you must manually fail over to another cluster.
- **Multi-cluster routing** automatically routes requests to the nearest cluster in an instance. If the cluster becomes unavailable, traffic automatically fails over to the nearest cluster that is available. Bigtable considers clusters in a single region to be equidistant, even though they are in different zones.

**Source:** Ex. 21, <https://cloud.google.com/bigtable/docs/app-profiles>, Page 1.

## Nodes

Each cluster in an instance has 1 or more nodes, which are compute resources that Bigtable uses to manage your data.

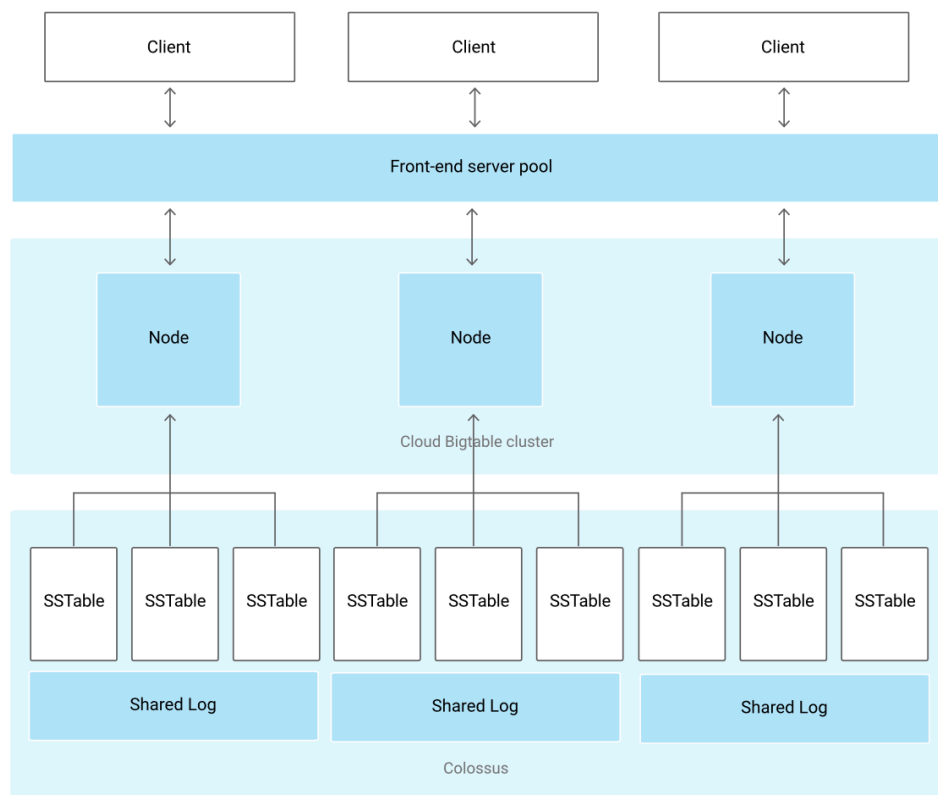
Behind the scenes, Bigtable splits all of the data in a table into separate *tablets*. Tablets are stored on disk, separate from the nodes but in the same zone as the nodes. A tablet is associated with a single node.

Each node is responsible for:

- Keeping track of specific tablets on disk.
- Handling incoming reads and writes for its tablets.
- Performing maintenance tasks on its tablets, such as periodic *compactions*.

**Source:** Ex. 22, <https://cloud.google.com/bigtable/docs/instances-clusters-nodes>, Page 3.

The following diagram shows a simplified version of Bigtable's overall architecture:



As the diagram illustrates, all client requests go through a frontend server before they are sent to a Bigtable node. (In the [original Bigtable paper](#), these nodes are called "tablet servers.") The nodes are organized into a Bigtable cluster, which belongs to a Bigtable instance, a container for the cluster.

**Source:** Ex. 20, <https://cloud.google.com/bigtable/docs/overview>, Page 3.



47. On information and belief, Google Colossus sends a location response message to the client in response to determining the one of the plurality of location servers includes the location information, the location response message comprising the location information. For example, as described above and as illustrated in the screenshots below, the tablet server (i.e., node) from the plurality of location servers is selected based on the routing technique defined by the user in the BigTable app profile. This metadata contains the list of SSTables that comprise a tablet. On the request to recover a tablet, a tablet server (node) obtains metadata from the Metadata table which stores the location information of the tablet. At the end of the SSTable a block index is stored which is used to locate the block. A lookup can be performed to find the appropriate block by performing a binary search in the in-memory index. The binary search returns the appropriate block including the location information from the disk.

sequence of SSTables. To recover a tablet, a tablet server

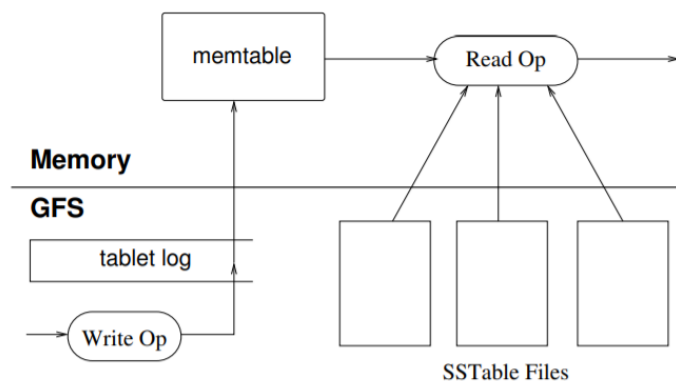


Figure 5: Tablet Representation

reads its metadata from the METADATA table. This metadata contains the list of SSTables that comprise a tablet and a set of redo points, which are pointers into any commit logs that may contain data for the tablet. The

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

key range. Internally, each SSTable contains a sequence of blocks (typically each block is 64KB in size, but this is configurable). A block index (stored at the end of the SSTable) is used to locate blocks; the index is loaded into memory when the SSTable is opened. A lookup can be performed with a single disk seek: we first find the appropriate block by performing a binary search in the in-memory index, and then reading the appropriate block from disk. Optionally, an SSTable can be com-

Fig. 1

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

48. On information and belief, Google Colossus sends a redirect message to the client in response to determining the one of the plurality of location servers fails to include the location information, the redirect message identifying which of the plurality of location servers includes the location information. As detailed above, Google Colossus uses BigTable to store tablet location information. As shown in the exemplary screenshot below, when the client determines that it does not know the location information from the location servers because the cached location is stale or incorrect, the location algorithm instructs the client to redirect, by moving up the tablet location hierarchy, and access another tablet server to find the location of the tablet. In this system, the cache miss response due to stale or incorrect location information is the redirect message. As described below, the location algorithm may take six round-trips in the case of stale cache entries to determine the location of the correct tablet server.

bytes in the metadata.

The client library caches tablet locations. If the client does not know the location of a tablet, or if it discovers that cached location information is incorrect, then it recursively moves up the tablet location hierarchy. If the client's cache is empty, the location algorithm requires three network round-trips, including one read from Chubby. If the client's cache is stale, the location algorithm could take up to six round-trips, because stale cache entries are only discovered upon misses (assuming that METADATA tablets do not move very frequently). Although tablet locations are stored in memory, so no GFS accesses are required, we further reduce this cost in the common case by having the client library prefetch tablet locations: it reads the metadata for more than one tablet whenever it reads the METADATA table.

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 5.

49. Based on the above, the Google Accused Products directly infringed at least claims 1 and 15 of the '170 Patent.

50. Kove is informed and believes, and thereon alleges, that Google has actively induced the infringement of the '170 Patent under 35 U.S.C. § 271(b) by actively inducing the infringing use of the Google Accused Products by third party users in the United States. Kove is informed and believes, and thereon alleges, that Google knew or should have known that its conduct would induce others to use the Google Accused Products in a manner that infringed the '170 Patent. Kove is informed and believes, and thereon alleges, that these third parties infringed the '170 Patent in violation of 35 U.S.C. § 271(a) by using the Google Accused Products.

51. For example, Google provides several support websites instructing third parties on the inner workings and functionality of the Google Accused Products, including, for example, materials informing users of *Colossus Under the Hood: A Peek Into Google's Scalable Storage*

*System*;<sup>20</sup> *Industry-Leading Reliability, Global Scale & Open Standards with Google Cloud Databases*;<sup>21</sup> *What is Cloud Spanner?*;<sup>22</sup> and *How Spanner and BigQuery Work Together to Handle Transactional and Analytical Workloads*.<sup>23</sup> These exemplary informative and instructional documents explain how the Google Accused Products can be used to store and retrieve data.

52. In addition, Google provides instructional documentation that explains how to use the Google Accused Products. For example, Google provides “Cloud Storage Documentation” that instructs users how to use the Google Accused Products to store data:<sup>24</sup>

Cloud Storage > Documentation > Guides

Was this helpful?  

## Upload objects

[Send feedback](#)

### On this page

Prerequisites

Upload an object to a bucket

What's next

This page shows you how to upload objects to your Cloud Storage bucket. An uploaded object consists of the data you want to store along with any associated metadata. For a conceptual overview, including how to choose the optimal upload method based on your file size, see [Uploads and downloads](#).

<sup>20</sup> Ex. 8. GOOGLE (Apr. 19, 2021), available at <https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system>.

<sup>21</sup> Ex. 23. GOOGLE (Mar. 14, 2022), available at <https://cloud.google.com/blog/products/databases/industry-leading-reliability-global-scale-open-standards-google-cloud-databases>.

<sup>22</sup> Ex. 9. GOOGLE (June 1, 2021), available at <https://cloud.google.com/blog/topics/developers-practitioners/what-cloud-spanner>.

<sup>23</sup> Ex. 24. GOOGLE (Mar. 10, 2022), available at <https://cloud.google.com/blog/topics/developers-practitioners/how-spanner-and-bigquery-work-together-handle-transactional-and-analytical-workloads>

<sup>24</sup> Ex. 25. GOOGLE, available at <https://cloud.google.com/storage/docs/uploading-objects>; Ex. 26, see also, e.g., *5-ish Ways to Get Your Data Into Cloud Storage*, GOOGLE (Mar. 5, 2021), available at <https://cloud.google.com/blog/topics/developers-practitioners/5-ish-ways-get-your-data-cloud-storage>.

53. Google provides various such resources to users of the Google Accused Products, including a series of instruction “Cloud Storage Bytes” videos<sup>25</sup> and “Google Cloud Platform Console Help.”<sup>26</sup> Further, Google provides “Training and Tutorials,” “Use Cases,” and “Code Samples” for users of the Google Accused Products.<sup>27</sup> Google also provides its users with tools, including client libraries,<sup>28</sup> software development kits,<sup>29</sup> and command line interface tools.<sup>30</sup> Additionally, Google educates and trains users of the Google Accused Products through its blog, where it regularly provides updates on “What’s New,” “Product News,” “Solutions & Technologies,” and various technical “Topics” relating to the Accused Products.<sup>31</sup>

54. Upon information and belief, Google contributorily infringed the ’170 Patent under 35 U.S.C. § 271(c) by importing, selling and/or offering to sell within the United States the Google Accused Products (or components thereof) that constitute a material part of the claimed invention and are not staple articles of commerce suitable for substantial non-infringing use. For example, the Google Accused Products, including Google Colossus and Google Spanner, are material, have no substantial non-infringing uses, and are known by Google to be especially made or especially adapted for use in the manner claimed in the ’170 Patent. Accordingly, Google contributorily infringed the ’170 Patent.

---

<sup>25</sup> Available at

<https://www.youtube.com/playlist?list=PLIivdWyY5sqJcBvDh5dfPoblLGhG1R1-O>.

<sup>26</sup> Ex. 27. *Google Cloud Platform Products: Cloud Storage*, GOOGLE, available at <https://support.google.com/cloud/answer/6250993?hl=en>.

<sup>27</sup> Ex. 28, See <https://cloud.google.com/spanner/docs>.

<sup>28</sup> Ex. 29. See, e.g., *Cloud Spanner Client Libraries*, GOOGLE, available at <https://cloud.google.com/spanner/docs/reference/libraries>.

<sup>29</sup> Ex. 30. See, e.g., *GCloud Spanner*, GOOGLE, available at <https://cloud.google.com/sdk/gcloud/reference/spanner>.

<sup>30</sup> See *id.*

<sup>31</sup> Ex. 31, See <https://cloud.google.com/blog/>.

55. As a result of Google's acts of infringement, Kove has suffered actual and consequential damages; however, Kove does not yet know the full extent of the infringement and its extent cannot be ascertained except through discovery and special accounting. To the fullest extent permitted by law, Kove seeks recovery of damages at least for reasonable royalties, unjust enrichment, and or benefits received by Google as a result of infringing the '170 Patent. Kove further seeks any other damages to which Kove is entitled under law or in equity.

56. In addition, Kove is entitled to recover reasonable and necessary attorneys' fees under applicable law.

#### **PRAYER FOR RELIEF**

WHEREFORE, Kove respectfully requests that this Court enter judgment in its favor and grant the following relief:

- A. A judgment that the Google Accused Products directly infringed the '170 patent;
- B. A ruling finding that this case is exceptional and awarding Kove its reasonable attorneys' fees under 35 U.S.C. § 285;
- C. A judgment and order requiring Google to pay Kove's damages in an amount adequate to compensate Kove for Google's infringement, but in no event less than a reasonable royalty under 35 U.S.C. § 284;
- D. A judgment and order requiring Google to pay Kove's costs of this action (including all disbursements);
- E. An order for accounting of damages;
- F. A judgment and order requiring Google to pay pre-judgment and post-judgment interest to the full extent allowed under the law; and,
- G. Such other and further relief as the Court may deem just and proper under the circumstances.

**COUNT II**  
**INFRINGEMENT OF THE '978 PATENT**

57. Kove incorporates by reference the foregoing paragraphs of this Complaint as if fully set forth herein.

58. On information and belief, Google has infringed the '978 Patent. Google directly infringed the '978 Patent under 35 U.S.C. §271(a) by making, using, selling, offering for sale, and/or importing in this District and into the United States products and/or methods covered by one or more claims of the '978 Patent, including, but not limited to, the Google Accused Products. As an example, the Google Accused Products infringed at least claim 17 of the '978 Patent.

59. Claim 17 is directed to a “method of scaling at least one of capacity and transaction rate capability in a location server in a system having a plurality of location servers for storing and retrieving location information.” Claim 17 recites, in its entirety:

17. A method of scaling at least one of capacity and transaction rate capability in a location server in a system having a plurality of location servers for storing and retrieving location information, wherein each of the plurality of location servers stores unique set of location information of an aggregate set of location information, the method comprising:

providing a transfer protocol configured to transport identifier and location information, the location information specifying the location of information related to the identifier;

storing location information formatted according to the transfer protocol at a first location server;

receiving an identifier and a location relevant to the identifier at the first location server;

storing the received location in a location store at the first data location server, the location store comprising a plurality of identifiers, each identifier associated with at least one location, wherein the received location is associated with the received identifier in the location store; and



transferring a portion of the identifiers and associated locations to a second data location server when a performance criterion of the first location server reaches a predetermined performance limit.

Ex. 6, Pg. 34.

**Google Colossus and Google Spanner infringed the '978 Patent.**

60. On information and belief, the Google Accused Products include a method of scaling at least one of capacity and transaction rate capability in a location server in a system having a plurality of location servers for storing and retrieving location information, wherein each of the plurality of location servers stores unique set of location information of an aggregate set of location information. For example, Google Spanner stores data in tablets stored on Google Colossus, which uses a NoSQL Database called Bigtable to store and retrieve the data location information on it.

Databases are part of virtually every application you run in your organization and great apps need great databases. This post is focused on one such great database—Cloud Spanner.

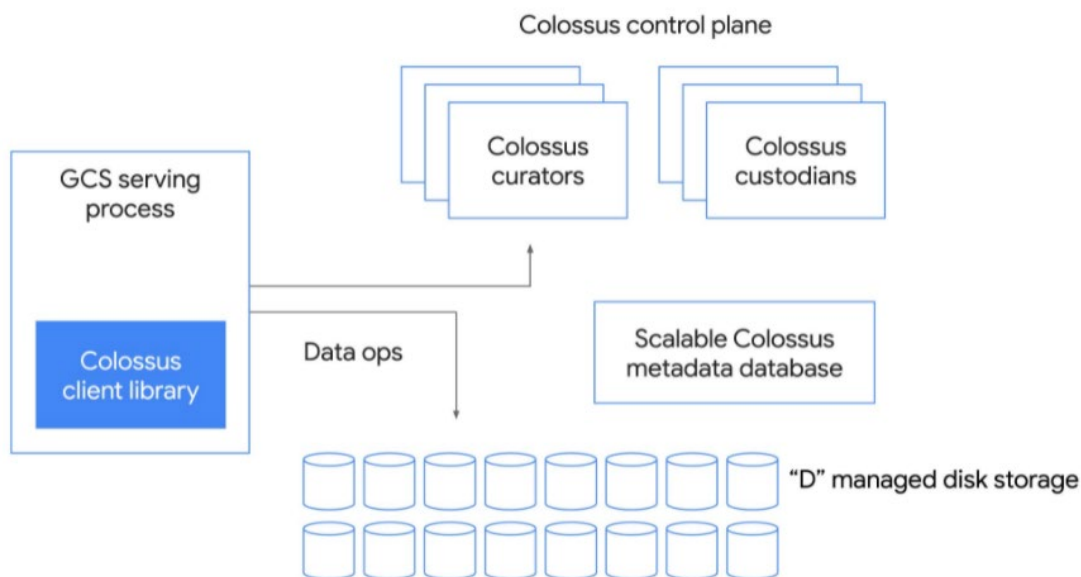
Cloud Spanner is the only enterprise-grade, globally-distributed, and strongly-consistent database service built for the cloud, specifically to combine the benefits of relational database structure with non-relational horizontal scale. It is a unique database that combines transactions, SQL queries, and relational structure with the scalability that you typically associate with non-relational or NoSQL databases.

**Source:** Ex. 9, *What is Cloud Spanner?*, GOOGLE (June 1, 2021), available at <https://cloud.google.com/blog/topics/developers-practitioners/what-cloud-spanner>.

which Spanner is more like a multiversion database than a key-value store. A tablet's state is stored in a set of B-tree-like files and a write-ahead log, all on a distributed file system called Colossus (the successor to the Google File System [Ghemawat et al.

**Source:** Ex. 18, <https://storage.googleapis.com/pub-tools-public-publication-data/pdf/65b514eda12d025585183a641b5a9e096a3c4be5.pdf>, Page 4.





Source: Ex. 8, <https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system>, Page 2.

61. On information and belief, the Google Accused Products provide a transfer protocol configured to transport identifier and location information, the location information specifying the location of information related to the identifier. For example, as illustrated in the screenshots below, Google Spanner consists of data structures called tablets which implement a “bag of mapping” keys to strings (i.e., identifiers). The tablets’ states, including the mapping strings, are stored in Google Colossus.

responsible for between 100 and 1000 instances of a data structure called a *tablet*. A tablet is similar to Bigtable’s tablet abstraction, in that it implements a bag of the following mappings.

(key:string, timestamp:int64) → string

Source: Ex. 18, <https://storage.googleapis.com/pub-tools-public-publication-data/pdf/65b514eda12d025585183a641b5a9e096a3c4be5.pdf>, Page 4.

which Spanner is more like a multiversion database than a key-value store. A tablet's state is stored in a set of B-tree-like files and a write-ahead log, all on a distributed file system called Colossus (the successor to the Google File System [Ghemawat et al.

**Source:** Ex. 18, <https://storage.googleapis.com/pub-tools-public-publication-data/pdf/65b514eda12d025585183a641b5a9e096a3c4be5.pdf>, Page 4.

62. Google Colossus provides a transfer protocol on a request to recover a tablet and its location information stored in a Metadata table. As discussed above, Google Colossus stores metadata in BigTable. As illustrated in the exemplary screenshots below, BigTable stores tablet location information, including a Metadata table that stores the location of a set of user tablets. The steps of storage and retrieval of location information stored in BigTable is a transfer protocol because all the requisite network communications occur as described further below. On the request to recover a tablet, the transfer protocol begins by a tablet server (node) reading its metadata from the Metadata table which stores the location information of the tablet. This metadata contains the list of SSTables.

#### **Metadata database**

Curators store file system metadata in Google's high-performance NoSQL database, [BigTable](#). The original motivation for building Colossus was to solve scaling limits we experienced with Google File System (GFS) when trying to accommodate metadata related to Search. Storing file metadata in BigTable allowed Colossus to scale up by over 100x over the largest GFS clusters.

**Source:** Ex. 8, <https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system>, Page 2.

## 5.1 Tablet Location

We use a three-level hierarchy analogous to that of a B<sup>+</sup>-tree [10] to store tablet location information (Figure 4).

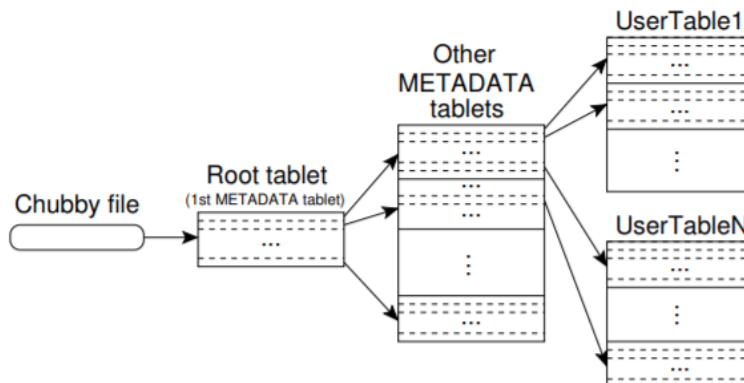


Figure 4: Tablet location hierarchy.

The first level is a file stored in Chubby that contains the location of the *root tablet*. The *root tablet* contains the location of all tablets in a special METADATA table. Each METADATA tablet contains the location of a set of user tablets. The *root tablet* is just the first tablet in the METADATA table, but is treated specially—it is never split—to ensure that the tablet location hierarchy has no more than three levels.

Source: Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

sequence of SSTables. To recover a tablet, a tablet server

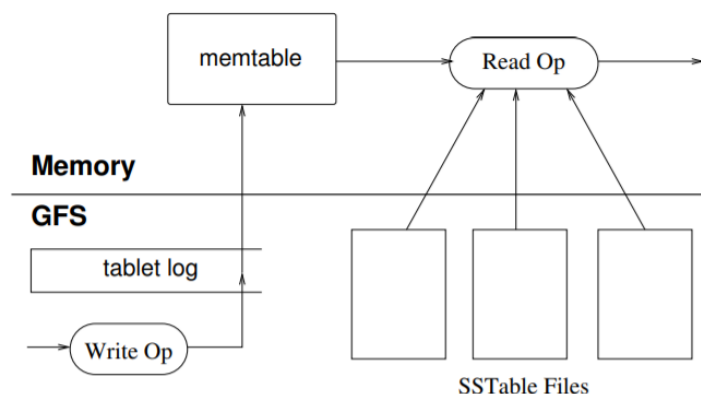


Figure 5: Tablet Representation

reads its metadata from the METADATA table. This meta-data contains the list of SSTables that comprise a tablet and a set of redo points, which are pointers into any commit logs that may contain data for the tablet. The

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en/archive/bigtable-osdi06.pdf>,

#### **Speeding up tablet recovery**

If the master moves a tablet from one tablet server to another, the source tablet server first does a minor compaction on that tablet. This compaction reduces recovery time by reducing the amount of uncompact state in the tablet server's commit log. After finishing this com-

Page 6.

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en/archive/bigtable-osdi06.pdf>,

Page 8.

63. The location information in Google Colossus specifies the location of information related to the identifier. Each SSTable contains a sequence of blocks and a map from keys to values (i.e., identifiers). At the end of the SSTable, a block index is stored which is used to locate the block (i.e., location information). A lookup can be performed with a single disk seek to find the appropriate block by performing a binary search in the in-memory index.

A Bigtable is a sparse, distributed, persistent multi-dimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

(row:string, column:string, time:int64) → string

**Source:** Ex. 19.

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 1.

key range. Internally, each SSTable contains a sequence of blocks (typically each block is 64KB in size, but this is configurable). A block index (stored at the end of the SSTable) is used to locate blocks; the index is loaded into memory when the SSTable is opened. A lookup can be performed with a single disk seek: we first find the appropriate block by performing a binary search in the in-memory index, and then reading the appropriate block from disk. Optionally, an SSTable can be com-

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

64. On information and belief, the Google Accused Products store location information formatted according to the transfer protocol at a first location server. For example, as illustrated in the screenshots below, BigTable uses Chubby services to keep track of tablet servers. BigTable stores tablet location information in tablet servers. The metadata table is stored as tablets which contain the locations of user tablets. *See, e.g.*, Ex. 19, Page 4 (“Each tablet server manages a set of tablets (typically we have somewhere between ten to a thousand tablets per tablet server). The tablet server handles read and write requests to the tablets that it has loaded, and also splits tablets that have grown too large.”).

Bigtable uses Chubby to keep track of tablet servers. When a tablet server starts, it creates, and acquires an exclusive lock on, a uniquely-named file in a specific Chubby directory. The master monitors this directory (the *servers directory*) to discover tablet servers. A tablet

Source: Ex.19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 5.

## 5.1 Tablet Location

We use a three-level hierarchy analogous to that of a B<sup>+</sup>-tree [10] to store tablet location information (Figure 4).

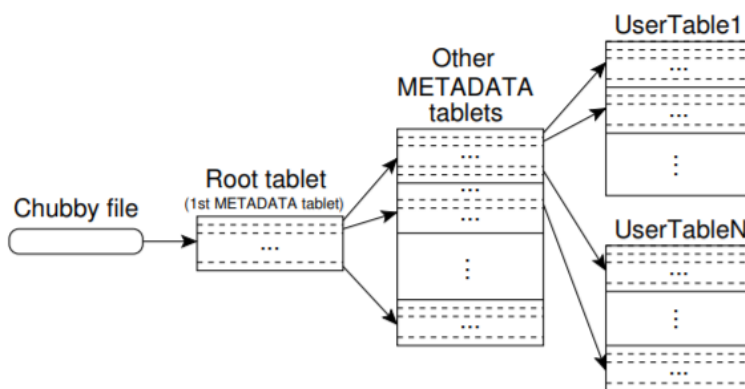


Figure 4: Tablet location hierarchy.

The first level is a file stored in Chubby that contains the location of the *root tablet*. The *root tablet* contains the location of all tablets in a special METADATA table. Each METADATA tablet contains the location of a set of user tablets. The *root tablet* is just the first tablet in the METADATA table, but is treated specially—it is never split—to ensure that the tablet location hierarchy has no more than three levels.

Source: Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

sequence of SSTables. To recover a tablet, a tablet server

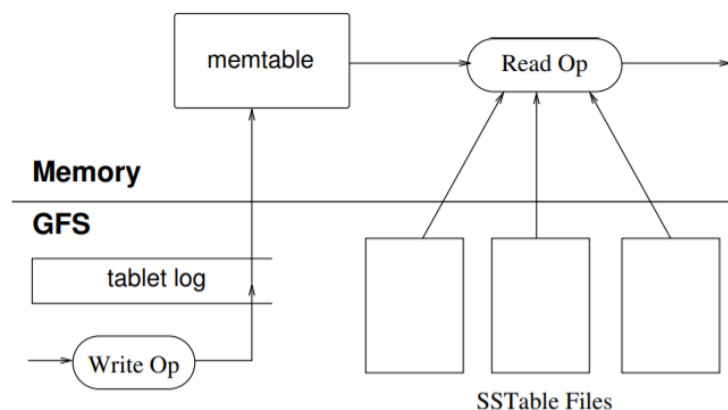


Figure 5: Tablet Representation

reads its metadata from the METADATA table. This metadata contains the list of SSTables that comprise a tablet and a set of redo points, which are pointers into any commit logs that may contain data for the tablet. The

Source: Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en/archive/bigtable-osdi06.pdf>,

Page 6.

65. This information is formatted according to the transfer protocol. For example, BigTable uses the SSTable format to store data, including the block index used to locate the block at the end of the SSTable.

The Google *SSTable* file format is used internally to store Bigtable data. An SSTable provides a persistent, ordered immutable map from keys to values, where both keys and values are arbitrary byte strings. Operations are provided to look up the value associated with a specified

Source: Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en/archive/bigtable-osdi06.pdf>,

Page 3.



key range. Internally, each SSTable contains a sequence of blocks (typically each block is 64KB in size, but this is configurable). A block index (stored at the end of the SSTable) is used to locate blocks; the index is loaded into memory when the SSTable is opened. A lookup can be performed with a single disk seek: we first find the appropriate block by performing a binary search in the in-memory index, and then reading the appropriate block from disk. Optionally, an SSTable can be completely mapped into memory, which allows us to perform lookups and scans without touching disk.

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

66. As a further example, Bigtable stores datasets with similar schemas in the same table. To retrieve data from the table (i.e. exemplary transfer protocol) BigTable uses Rowkeys. Rowkeys include a plurality of identifiers separated by a delimiter, such as a colon, slash, or hash symbol. By way of one example, if the application tracks mobile device data, a row key that consists of a plurality of identifiers such as device type, device ID, and the day the data is recorded.

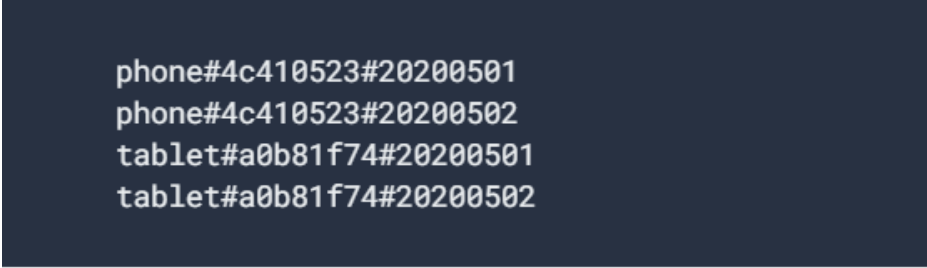
#### Tables

**Store datasets with similar schemas in the same table**, rather than in separate tables.

In other database systems, you might choose to store data in multiple tables based on the subject and number of columns. In Bigtable, however, it is usually better to store all your data in one big table. You can assign a unique row key prefix to use for each dataset, so that Bigtable stores the related data in a contiguous range of rows that you can then query by row key prefix.

**Source:** Ex. 32, <https://cloud.google.com/bigtable/docs/schema-design>, Page 3.

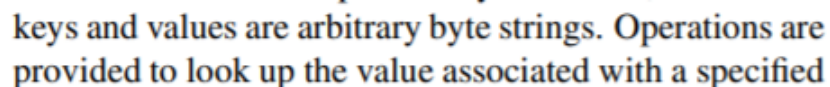




```
phone#4c410523#20200501
phone#4c410523#20200502
tablet#a0b81f74#20200501
tablet#a0b81f74#20200502
```

**Source:** Ex. 32, <https://cloud.google.com/bigtable/docs/schema-design>, Page 6.

67. On information and belief, the Google Accused Products receive an identifier and a location relevant to the identifier at the first location server. For example, as illustrated in the screenshots below, an SSTable consists of a map from keys to values, where values serve as identifiers. For example, a lookup operation which contains the key can be performed by using the key to iterate over the key/value pairs to find an associated value. The SSTable also contains a block index, which is stored and used to locate the blocks of the SSTable. This information is received, for example, when tablet data, including that in the SSTable, is transferred between tablet servers.

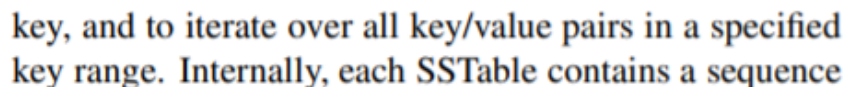


keys and values are arbitrary byte strings. Operations are  
provided to look up the value associated with a specified

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 3.



key, and to iterate over all key/value pairs in a specified  
key range. Internally, each SSTable contains a sequence

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

key range. Internally, each SSTable contains a sequence of blocks (typically each block is 64KB in size, but this is configurable). A block index (stored at the end of the SSTable) is used to locate blocks; the index is loaded into memory when the SSTable is opened. A lookup can be performed with a single disk seek: we first find the appropriate block by performing a binary search in the in-memory index, and then reading the appropriate block from disk. Optionally, an SSTable can be completely mapped into memory, which allows us to perform lookups and scans without touching disk.

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

#### **Speeding up tablet recovery**

If the master moves a tablet from one tablet server to another, the source tablet server first does a minor compaction on that tablet. This compaction reduces recovery time by reducing the amount of uncompact state in the tablet server's commit log. After finishing this com-

Page 4.

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 8.

68. On information and belief, the Google Accused Products store the received location in a location store at the first data location server, the location store comprising a plurality of identifiers, each identifier associated with at least one location, wherein the received location is associated with the received identifier in the location store. For example, as illustrated in the screenshots below, SSTable containing the block index (location information) is stored in the tablet servers which store Metadata tablets.

key range. Internally, each SSTable contains a sequence of blocks (typically each block is 64KB in size, but this is configurable). A block index (stored at the end of the SSTable) is used to locate blocks; the index is loaded into memory when the SSTable is opened. A lookup can be performed with a single disk seek: we first find the appropriate block by performing a binary search in the in-memory index, and then reading the appropriate block from disk. Optionally, an SSTable can be completely mapped into memory, which allows us to perform lookups and scans without touching disk.

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

### **Speeding up tablet recovery**

If the master moves a tablet from one tablet server to another, the source tablet server first does a minor compaction on that tablet. This compaction reduces recovery time by reducing the amount of uncompact state in the tablet server's commit log. After finishing this com-

**Source:** Ex., 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 8.

sequence of SSTables. To recover a tablet, a tablet server

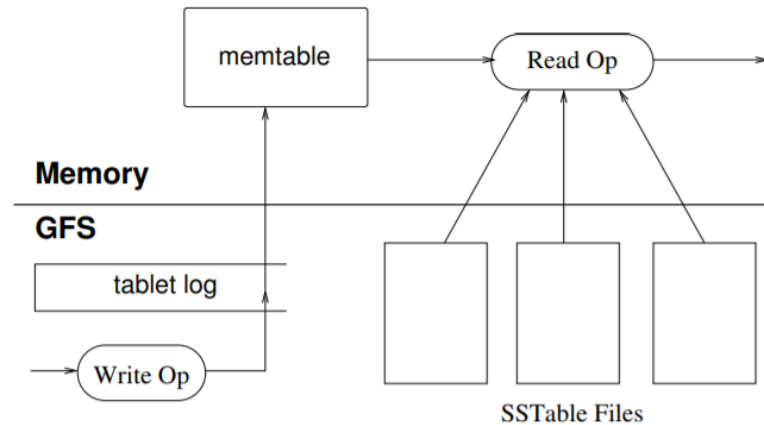


Figure 5: Tablet Representation

reads its metadata from the METADATA table. This metadata contains the list of SSTables that comprise a tablet and a set of redo points, which are pointers into any commit logs that may contain data for the tablet. The

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 6.

## 5.1 Tablet Location

We use a three-level hierarchy analogous to that of a B<sup>+</sup>-tree [10] to store tablet location information (Figure 4).

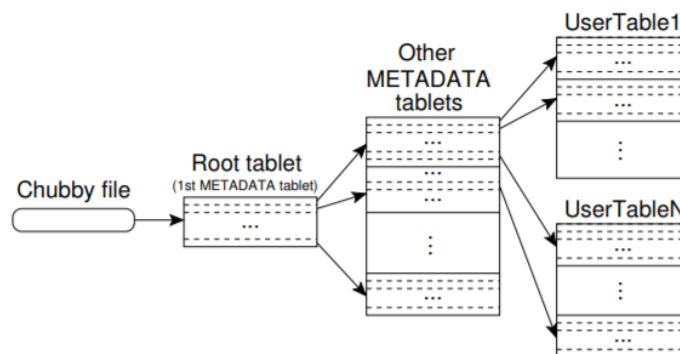


Figure 4: Tablet location hierarchy.

The first level is a file stored in Chubby that contains the location of the *root tablet*. The *root tablet* contains the location of all tablets in a special METADATA table. Each METADATA tablet contains the location of a set of user tablets. The *root tablet* is just the first tablet in the METADATA table, but is treated specially—it is never split—to ensure that the tablet location hierarchy has no more than three levels.

Source: Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

69. In BigTable, the Metadata table serves as a location store by storing the location information of the tablet. BigTable stores datasets with similar schemas in the same table. To retrieve data from the table, BigTable uses Rowkeys. Rowkeys include a plurality of identifiers separated by a delimiter, such as a colon, slash, or hash symbol. By way of one example, if the application tracks mobile device location data, a Rowkey that consists of a plurality of identifiers such as device location, device ID, and the day the data is recorded.

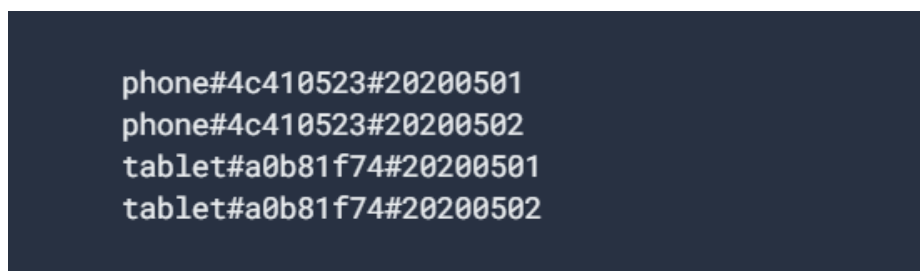
## Tables

**Store datasets with similar schemas in the same table**, rather than in separate tables.

In other database systems, you might choose to store data in multiple tables based on the subject and number of columns. In Bigtable, however, it is usually better to store all your data in one big table. You can assign a unique row key prefix to use for each dataset, so that Bigtable stores the related data in a contiguous range of rows that you can then query by row key prefix.

**Source:** Ex. 19, <https://cloud.google.com/bigtable/docs/schema-design>, Page 3.

### Citation 1: Rowkeys



```
phone#4c410523#20200501
phone#4c410523#20200502
tablet#a0b81f74#20200501
tablet#a0b81f74#20200502
```

Fig. 25

**Source:** Ex. 19, <https://cloud.google.com/bigtable/docs/schema-design>, Page 6.

70. On information and belief, the Google Accused Products transfer a portion of the identifiers and associated locations to a second data location server when a performance criterion of the first location server reaches a predetermined performance limit. For example, as illustrated in the screenshots below, Google Colossus uses Custodians to ensure the reliability of storage hardware. By way of further example, as illustrated in the screenshots below, BigTable implements systems that “balanc[e] tablet-server load” and each “table server . . . also splits tablets that have grown to large.” Balancing server loads (i.e., “load balancing”), for example, is the process of moving information, including location information stored in BigTable, from one disk drive, or server, to another disk drive when a drive reaches a predetermined performance limit, such as reaching a capacity limit or failure.

The Bigtable implementation has three major components: a library that is linked into every client, one master server, and many tablet servers. Tablet servers can be

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

The master is responsible for assigning tablets to tablet servers, detecting the addition and expiration of tablet servers, balancing tablet-server load, and garbage collection of files in GFS. In addition, it handles schema changes such as table and column family creations.

Each tablet server manages a set of tablets (typically we have somewhere between ten to a thousand tablets per tablet server). The tablet server handles read and write requests to the tablets that it has loaded, and also splits tablets that have grown too large.

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

ery. When a tablet server dies, the tablets that it served will be moved to a large number of other tablet servers: each server typically loads a small number of the original server's tablets. To recover the state for a tablet,

**Source:** Ex. 19.

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 7.

Colossus uses a set of programs Google's engineers refer to as Custodians to ensure the reliability of storage hardware. When one of the disk drives inside a storage system fails, the Custodians can reassemble the lost information from the data remaining in the drives that are still operational. The programs also perform a variety of other tasks to increase the durability of Google's storage environments, which can scale to multiple exabytes and tens of thousands of machines, according to Hildebrand and Serenyi.



**Source:** Ex. 33, <https://siliconangle.com/2021/04/19/google-shares-technical-overview-exabyte-scale-colossus-file-system/>, Page 2-3.

The data itself flows directly between the client application and the "D" file servers, which are network-attached disks. Custodians operate on the file serves as well, playing "a key role in maintaining the durability and availability of data as well as overall efficiency, handling tasks like disk space balancing and RAID reconstruction."

**Source:** Ex. 34, <https://www.infoq.com/news/2021/04/google-colossus/>, Page 2.

Each node in the cluster handles a subset of the requests to the cluster. By adding nodes to a cluster, you can increase the number of simultaneous requests that the cluster can handle. Adding nodes also increases the maximum throughput for the cluster. If you enable replication by adding additional clusters, you can also send different types of traffic to different clusters. Then if one cluster becomes unavailable, you can fail over to another cluster.

A Bigtable table is sharded into blocks of contiguous rows, called *tablets*, to help balance the workload of queries. (Tablets are similar to HBase regions.) Tablets are stored on Colossus, Google's file system, in SSTable format. An SSTable provides a persistent, ordered immutable map from keys to values, where both keys and values are arbitrary byte strings. Each tablet is associated with a specific Bigtable node. In addition to the SSTable files, all writes are stored in Colossus's shared log as soon as they are acknowledged by Bigtable, providing increased durability.

Importantly, data is never stored in Bigtable nodes themselves; each node has pointers to a set of tablets that are stored on Colossus. As a result:

- Rebalancing tablets from one node to another happens quickly, because the actual data is not copied. Bigtable simply updates the pointers for each node.
- Recovery from the failure of a Bigtable node is fast, because only metadata must be migrated to the replacement node.
- When a Bigtable node fails, no data is lost.

**Source:** Ex. 20, <https://cloud.google.com/bigtable/docs/overview>.

71. Based on the above, the Google Accused Products directly infringed at least claim 17 of the '978 Patent.

72. Kove is informed and believes, and thereon alleges, that Google has actively induced the infringement of the '978 Patent under 35 U.S.C. § 271(b) by actively inducing the infringing use of the Google Accused Products by third party users in the United States. Kove is informed and believes, and thereon alleges, that Google knew or should have known that its



conduct would induce others to use the Google Accused Products in a manner that infringed the '978 Patent. Kove is informed and believes, and thereon alleges, that these third parties infringed the '978 Patent in violation of 35 U.S.C. § 271(a) by using the Google Accused Products.

73. Upon information and belief, Google contributorily infringed the '978 Patent under 35 U.S.C. § 271(c) by importing, selling and/or offering to sell within the United States the Google Accused Products (or components thereof) that constitute a material part of the claimed invention and are not staple articles of commerce suitable for substantial non-infringing use. For example, the Google Accused Products, including Google Colossus and Google Spanner, are material, have no substantial non-infringing uses, and are known by Google to be especially made or especially adapted for use in the manner claimed in the '978 Patent. Accordingly, Google contributorily infringed the '978 Patent.

74. As a result of Google's acts of infringement, Kove has suffered actual and consequential damages; however, Kove does not yet know the full extent of the infringement and its extent cannot be ascertained except through discovery and special accounting. To the fullest extent permitted by law, Kove seeks recovery of damages at least for reasonable royalties, unjust enrichment, and or benefits received by Google as a result of infringing the '978 Patent. Kove further seeks any other damages to which Kove is entitled under law or in equity.

75. In addition, Kove is entitled to recover reasonable and necessary attorneys' fees under applicable law.

#### **PRAYER FOR RELIEF**

WHEREFORE, Kove respectfully requests that this Court enter judgment in its favor and grant the following relief:

A. A judgment that the Google Accused Products directly infringed the '978 patent;

- B. A ruling finding that this case is exceptional and awarding Kove its reasonable attorneys' fees under 35 U.S.C. § 285;
- C. A judgment and order requiring Google to pay Kove's damages in an amount adequate to compensate Kove for Google's infringement, but in no event less than a reasonable royalty under 35 U.S.C. § 284;
- D. A judgment and order requiring Google to pay Kove's costs of this action (including all disbursements);
- E. An order for accounting of damages;
- F. A judgment and order requiring Google to pay pre-judgment and post-judgment interest to the full extent allowed under the law; and,
- G. Such other and further relief as the Court may deem just and proper under the circumstances.

**COUNT III**  
**INFRINGEMENT OF THE '640 PATENT**

76. Kove incorporates by reference the foregoing paragraphs of this Complaint as if fully set forth herein.

77. On information and belief, Google has infringed the '640 Patent. Google directly infringed the '640 Patent under 35 U.S.C. §271(a) by making, using, selling, offering for sale, and/or importing in this District and into the United States products and/or methods covered by one or more claims of the '640 Patent, including, but not limited to, the Google Accused Products. As an example, the Google Accused Products infringed at least claim 10 of the '640 Patent.

78. Claim 10 is directed to a "method for retrieving data location information for data stored in a distributed network." For example, claim 10 recites "receiving . . . a data query for

retrieving data associated with an identifier string,” “transmitting a data location request,” “transmitting a redirect message” “if the first data location server does not possess the location string,” “calculating the location of the second location server,” and “transmitting the data query from the first client to the second data location server.

79. Claim 10, in its entirety, recites:

10. A method for retrieving data location information for data stored in a distributed network, comprising the steps of:
  - a) receiving at a first client a data query for retrieving data associated with an identification string, wherein the data is stored at a data repository in the distributed network and wherein a location string associated with the identification string of the data is stored in at least one of a plurality of data location servers;
  - b) transmitting a data location request from the first client to a first data location server to retrieve the location string associated with the identification string in the data query, the data location request including the identification string;
  - c) if the first data location server does not possess the location string, transmitting a redirect message to the first client, the redirect message containing information for use by the first client to calculate a location of a second data location server, wherein the second data location server contains the location string;
  - d) calculating the location of the second data location server at the first client; and
  - e) transmitting the data query from the first client to the second data location server.

Ex. 7, Pg. 26.

**Google Spanner and Google Colossus infringed the '640 Patent.**

80. On information and belief, the Google Accused Products include a method for retrieving data location information for data stored in a distributed network. For example, as shown in the screenshots below, Google Colossus is a distributed metadata system and Google Spanner is a globally distributed database system.

## Colossus in a nutshell

Now, let's take a closer look at how Colossus works.

But first, a little background on Colossus:

- It's the next-generation of the GFS.
- Its design enhances storage scalability and improves availability to handle the massive growth in data needs of an ever-growing number of applications.
- Colossus introduced a distributed metadata model that delivered a more scalable and highly available metadata subsystem.

**Source:** Ex. 8, *Colossus Under the Hood: A Peek Into Google's Scalable Storage System*, GOOGLE (Apr. 19, 2021), available at <https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system>.

Databases are part of virtually every application you run in your organization and great apps need great databases. This post is focused on one such great database—Cloud Spanner.

Cloud Spanner is the only enterprise-grade, globally-distributed, and strongly-consistent database service built for the cloud, specifically to combine the benefits of relational database structure with non-relational horizontal scale. It is a unique database that combines transactions, SQL queries, and relational structure with the scalability that you typically associate with non-relational or NoSQL databases.

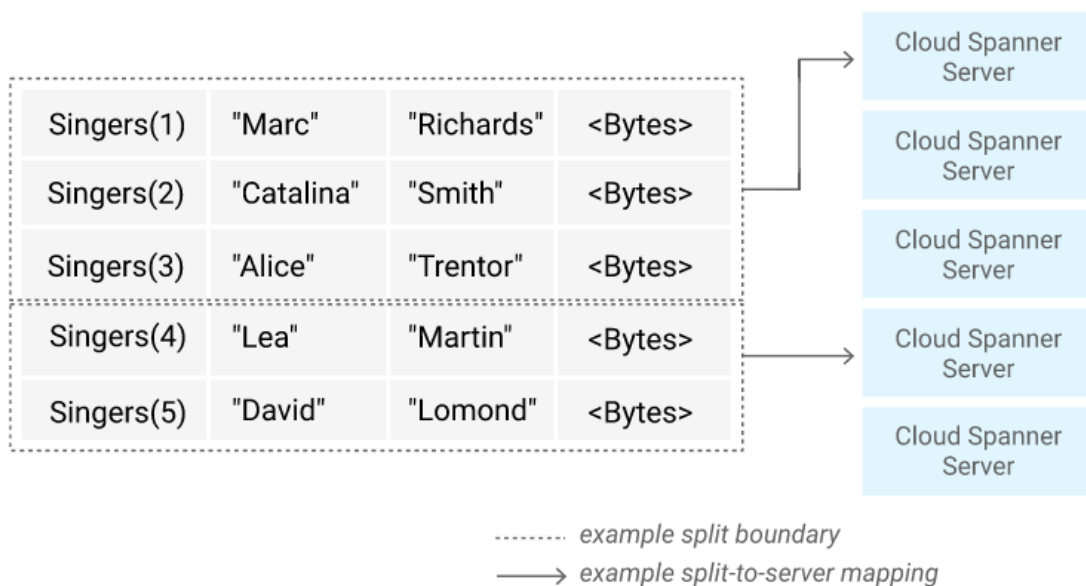
**Source:** Ex. 9, *What is Cloud Spanner?*, GOOGLE (June 1, 2021), available at <https://cloud.google.com/blog/topics/developers-practitioners/what-cloud-spanner>. Ex. 9.

81. On information and belief, the Google Accused Products receive at a first client a data query for retrieving data associated with an identification string, wherein the data is stored at a data repository in the distributed network and wherein a location string associated with the identification string of the data is stored in at least one of a plurality of data location servers.

82. For example, Google Spanner is a distributed database that replicates user data across multiple zones. Google Spanner divides user data into chunks called splits that are assigned to different data repository servers based on primary key value. As explained above, the key-value mappings and tablet information of Spanner are stored on Google Colossus.

database grows, Cloud Spanner divides your data into chunks called "splits", where individual splits can move independently from each other and get assigned to different servers, which can be in different physical locations. A split holds a range of contiguous rows. The start and end keys of this range are called "split boundaries". Cloud Spanner

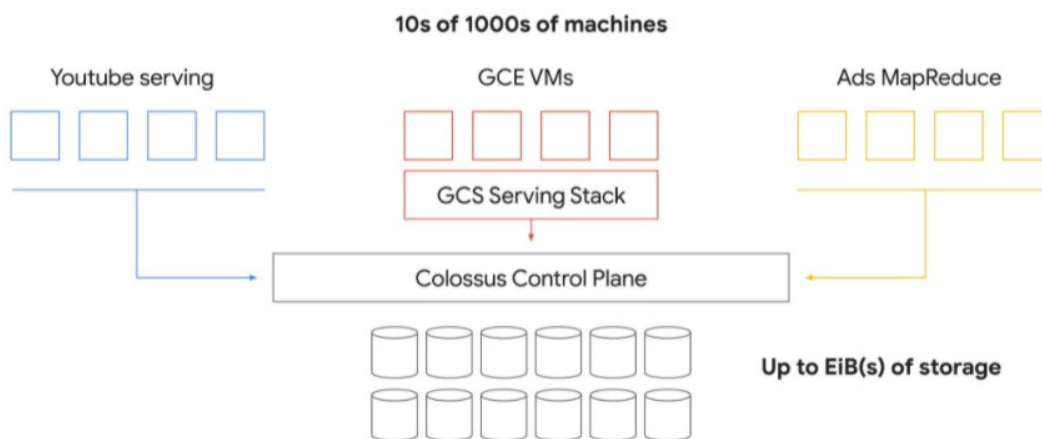
**Source:** Ex. 16, <https://cloud.google.com/spanner/docs/schema-and-data-model>, Page 3.



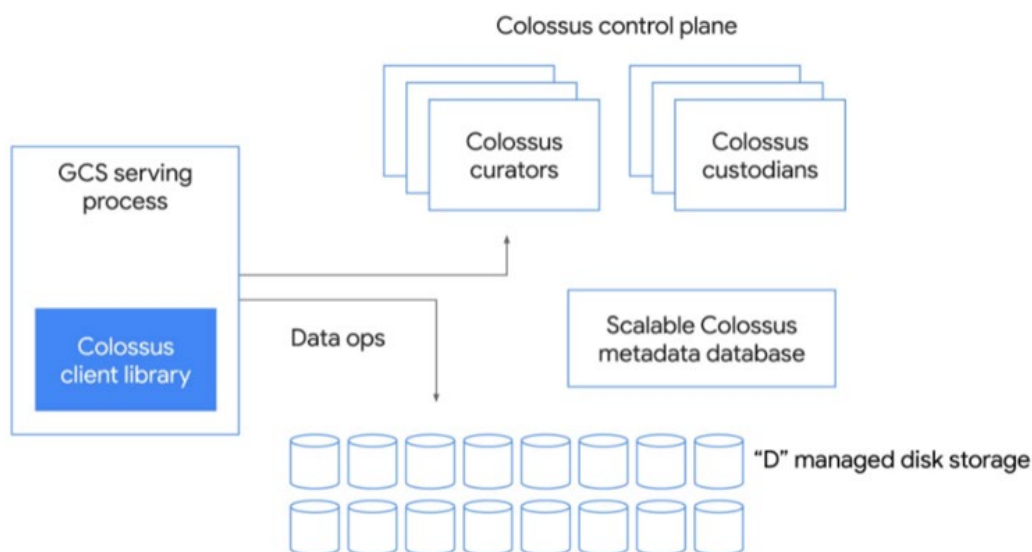
**Source:** Ex. 16, <https://cloud.google.com/spanner/docs/schema-and-data-model>, Page 5.

83. Google Colossus receives at a first client a data query for retrieving data associated with an identification string, wherein the data is stored at a data repository in the distributed network and wherein a location string associated with the identification string of the data is stored in at least one of a plurality of data location servers. For example, as illustrated in the screenshots below, the Google Colossus Control Plane interacts with clients, including Google's most popular and valuable products and services, such as YouTube, GCE, and Google Ads.

## Typical cluster



Source: Ex. 8, <https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system>, Page 3.



Source: Ex. 8, <https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system>, Page 2.

84. These clients request data and metadata, such as URLs, that are stored in the rows and columns of Google BigTable, contained in the Colossus Control Plane. This metadata is

associated with a key-value identification string, and as part of the data associated with the URLs is stored in the “D” managed disk storage data repository.

## 2 Data Model

A Bigtable is a sparse, distributed, persistent multi-dimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

$(\text{row: string, column: string, time: int64}) \rightarrow \text{string}$

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 1.

85. BigTable also stores location information of the identification data in the form of block indices stored in an SSTable. Each SSTable stores a Block index at the end of it. The Block indices (i.e. location strings) are used to locate Blocks.

key range. Internally, each SSTable contains a sequence of blocks (typically each block is 64KB in size, but this is configurable). A block index (stored at the end of the SSTable) is used to locate blocks; the index is loaded into memory when the SSTable is opened. A lookup can be performed with a single disk seek: we first find the appropriate block by performing a binary search in the in-memory index, and then reading the appropriate block from disk. Optionally, an SSTable can be completely mapped into memory, which allows us to perform lookups and scans without touching disk.

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

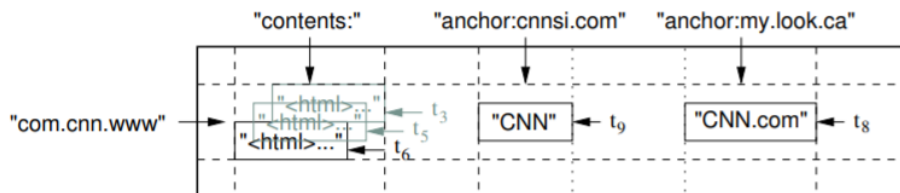


Figure 1: A slice of an example table that stores Web pages. The row name is a reversed URL. The `contents` column family contains the page contents, and the `anchor` column family contains the text of any anchors that reference the page. CNN's home page is referenced by both the Sports Illustrated and the MY-look home pages, so the row contains columns named `anchor:cnnsi.com` and `anchor:my.look.ca`. Each anchor cell has one version; the contents column has three versions, at timestamps  $t_3$ ,  $t_5$ , and  $t_6$ .

Source: Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 2.

#### D File Servers

Colossus also minimizes the number of hops for data on the network. Data flows directly between clients and "D" file servers (our network attached disks).

Source: Ex. 8, <https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system>, Page 2.

86. A BigTable cluster consists of Tables. Each table consists of several tablets. The root tablet stores the location of all tablets in a special Metadata table. There exist other Metadata tablets that store the location of a set of User Tablets. One major component of BigTable is a tablet server that consists of several Tablets. Hence, there exist several Metadata Tablet Servers (i.e., data location servers) that contain location information of the various user tablets.



A Bigtable cluster stores a number of tables. Each table consists of a set of tablets, and each tablet contains all data associated with a row range. Initially, each table consists of just one tablet. As a table grows, it is automatically split into multiple tablets, each approximately 100-200 MB in size by default.

Source: Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

### 5.1 Tablet Location

We use a three-level hierarchy analogous to that of a  $B^+$ -tree [10] to store tablet location information (Figure 4).

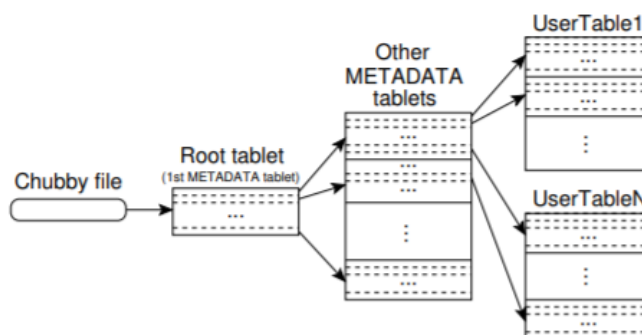


Figure 4: Tablet location hierarchy.

The first level is a file stored in Chubby that contains the location of the *root tablet*. The *root tablet* contains the location of all tablets in a special METADATA table. Each METADATA tablet contains the location of a set of user tablets. The *root tablet* is just the first tablet in the METADATA table, but is treated specially—it is never split—to ensure that the tablet location hierarchy has no more than three levels.

Source: Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

Each tablet server manages a set of tablets (typically we have somewhere between ten to a thousand tablets per tablet server). The tablet server handles read and write requests to the tablets that it has loaded, and also splits tablets that have grown too large.

Source: Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 4.

87. On information and belief, Google transmits a data location request from the first client to a first data location server to retrieve the location string associated with the identification string in the data query, the data location request including the identification string. As illustrated in the screenshots below, in Google Colossus, a client requests information of data from BigTable, for example through a lookup request. For example, BigTable stores data in SSTable entities, which contain key-value (i.e. identification string) stores. The SSTable also contains a Block index, and BigTable performs a lookup operation by first finding the appropriate Block, and then reading the appropriate block from the disk.

Bigtable uses Chubby to keep track of tablet servers. When a tablet server starts, it creates, and acquires an exclusive lock on, a uniquely-named file in a specific Chubby directory. The master monitors this directory (the *servers directory*) to discover tablet servers. A tablet

Source: Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 5.

keys and values are arbitrary byte strings. Operations are provided to look up the value associated with a specified

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en/archive/bigtable-osdi06.pdf>,

Page 3.

key, and to iterate over all key/value pairs in a specified key range. Internally, each SSTable contains a sequence

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en/archive/bigtable-osdi06.pdf>,

Page 4.

key range. Internally, each SSTable contains a sequence of blocks (typically each block is 64KB in size, but this is configurable). A block index (stored at the end of the SSTable) is used to locate blocks; the index is loaded into memory when the SSTable is opened. A lookup can be performed with a single disk seek: we first find the appropriate block by performing a binary search in the in-memory index, and then reading the appropriate block from disk. Optionally, an SSTable can be completely mapped into memory, which allows us to perform lookups and scans without touching disk.

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en/archive/bigtable-osdi06.pdf>,

Page 4.

88. On information and belief, if the first data location server does not possess the location string, the Google Accused Products transmit a redirect message to the first client, the redirect message containing information for use by the first client to calculate a location of a second data location server, wherein the second data location server contains the location string. For

example, as illustrated in the screenshot below, the client library caches the location of a tablet. When the cached location at the client is stale or incorrect, the client detects a miss, which is the redirect message, and the location algorithm redirects to another tablet location, utilizing a location algorithm which calculates a location of a second data location server to determine the location of the correct tablet server. The location algorithm redirects by instructing the client to move up the tablet hierarchy.

The client library caches tablet locations. If the client does not know the location of a tablet, or if it discovers that cached location information is incorrect, then it recursively moves up the tablet location hierarchy. If the client's cache is empty, the location algorithm requires three network round-trips, including one read from Chubby. If the client's cache is stale, the location algorithm could take up to six round-trips, because stale cache entries are only discovered upon misses (assuming that METADATA tablets do not move very frequently). Although tablet locations are stored in memory, so no GFS accesses are required, we further reduce this cost in the common case by having the client library prefetch tablet locations: it reads the metadata for more than one tablet whenever it reads the METADATA table.

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 5.

89. On information and belief, the Google Accused Products calculate the location of the second data location server at the first client. For example, as illustrated in the screenshot below, the location algorithm calculates the location of the second tablet server to access the location of the required data. The location algorithm determines the tablet location using three round trips in the case of empty cache and six round trips in the case of misses.

The client library caches tablet locations. If the client does not know the location of a tablet, or if it discovers that cached location information is incorrect, then it recursively moves up the tablet location hierarchy. If the client's cache is empty, the location algorithm requires three network round-trips, including one read from Chubby. If the client's cache is stale, the location algorithm could take up to six round-trips, because stale cache entries are only discovered upon misses (assuming that METADATA tablets do not move very frequently). Although tablet locations are stored in memory, so no GFS accesses are required, we further reduce this cost in the common case by having the client library prefetch tablet locations: it reads the metadata for more than one tablet whenever it reads the METADATA table.

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 5.

90. On information and belief, the Google Accused Products transmit the data query from the first client to the second data location server. For example, as illustrated in the exemplary screenshots below, once the second tablet server is discovered, the location of that server is returned to the client to complete the request and also cached by obtaining a lock on the server directory (i.e. the data query is transmitted from the first client to the second data location server).

Bigtable uses Chubby to keep track of tablet servers. When a tablet server starts, it creates, and acquires an exclusive lock on, a uniquely-named file in a specific Chubby directory. The master monitors this directory (the *servers directory*) to discover tablet servers. A tablet

**Source:** Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 5.



The client library caches tablet locations. If the client does not know the location of a tablet, or if it discovers that cached location information is incorrect, then it recursively moves up the tablet location hierarchy. If the client's cache is empty, the location algorithm requires three network round-trips, including one read from Chubby. If the client's cache is stale, the location algorithm could take up to six round-trips, because stale cache entries are only discovered upon misses (assuming that METADATA tablets do not move very frequently). Although tablet locations are stored in memory, so no GFS accesses are required, we further reduce this cost in the common case by having the client library prefetch tablet locations: it reads the metadata for more than one tablet whenever it reads the METADATA table.

Source: Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf>,

Page 5.

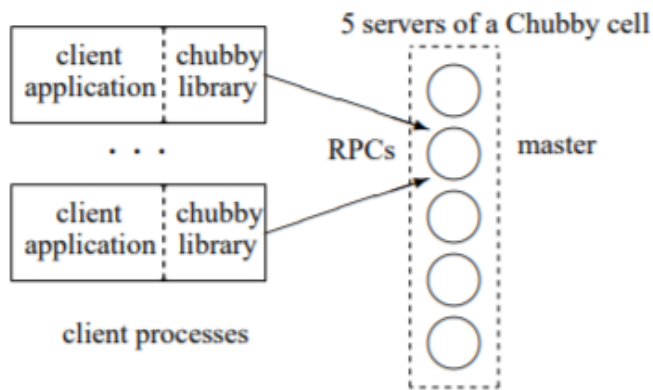


Figure 1: System structure

Source: Ex. 19,

<https://static.googleusercontent.com/media/research.google.com/en//archive/chubby-osdi06.pdf>,

Page 3.

91. Based on the above, the Google Accused Products directly infringed at least claim 10 of the '640 Patent.

92. Kove is informed and believes, and thereon alleges, that Google has actively induced the infringement of the '640 Patent under 35 U.S.C. § 271(b) by actively inducing the infringing use of the Google Accused Products by third party users in the United States. Kove is informed and believes, and thereon alleges, that Google knew or should have known that its conduct would induce others to use the Google Accused Products in a manner that infringed the '640 Patent. Kove is informed and believes, and thereon alleges, that these third parties infringed the '640 Patent in violation of 35 U.S.C. § 271(a) by using the Google Accused Products.

93. Upon information and belief, Google contributorily infringed the '640 Patent under 35 U.S.C. § 271(c) by importing, selling and/or offering to sell within the United States the Google Accused Products (or components thereof) that constitute a material part of the claimed invention and are not staple articles of commerce suitable for substantial non-infringing use. For example, the Google Accused Products, including Google Colossus and Google Spanner, are material, have no substantial non-infringing uses, and are known by Google to be especially made or especially adapted for use in the manner claimed in the '640 Patent. Accordingly, Google contributorily infringed the '640 Patent.

94. As a result of Google's acts of infringement, Kove has suffered actual and consequential damages; however, Kove does not yet know the full extent of the infringement and its extent cannot be ascertained except through discovery and special accounting. To the fullest extent permitted by law, Kove seeks recovery of damages at least for reasonable royalties, unjust enrichment, and/or benefits received by Google as a result of infringing the '640 Patent. Kove further seeks any other damages to which Kove is entitled under law or in equity.

95. In addition, Kove is entitled to recover reasonable and necessary attorneys' fees under applicable law.

**PRAYER FOR RELIEF**

WHEREFORE, Kove respectfully requests that this Court enter judgment in its favor on Count III of its Complaint and grant the following relief:

- A. A judgment that the Google Accused Products directly infringed the '640 patent;
- B. A ruling finding that this case is exceptional and awarding Kove its reasonable attorneys' fees under 35 U.S.C. § 285;
- C. A judgment and order requiring Google to pay Kove's damages in an amount adequate to compensate Kove for Google's infringement, but in no event less than a reasonable royalty under 35 U.S.C. § 284;
- D. A judgment and order requiring Google to pay Kove's costs of this action (including all disbursements);
- E. An order for accounting of damages;
- F. A judgment and order requiring Google to pay pre-judgment and post-judgment interest to the full extent allowed under the law; and,
- G. Such other and further relief as the Court may deem just and proper under the circumstances.

**DEMAND FOR JURY TRIAL**

Kove hereby demands a jury trial on its claims for patent infringement.

Date: November 14, 2023

Respectfully Submitted,

**KOVE IO, INC.,**

Plaintiff.

/s/ Khue V. Hoang



Khue Hoang (NY#4131850)  
Jaime F. Cardenas-Navia (NY#5249248)  
**REICHMAN JORGENSEN LEHMAN &  
FELDBERG LLP**  
400 Madison Avenue, Suite 14D  
New York, NY 10017  
(646) 921-1474  
khoang@reichmanjorgensen.com  
jcardenas-navia@reichmanjorgensen.com

Gina H. Cremona (*pro hac vice*)  
**REICHMAN JORGENSEN LEHMAN &  
FELDBERG LLP**  
100 Marine Parkway, Suite 300  
Redwood Shores, CA 94065  
(650) 623-1401  
gcremona@reichmanjorgensen.com

Maxwell A. Kling (Bar No. 6327171)  
**HORWOOD MARCUS & BERK  
CHARTERED**  
500 West Madison Street, Suite 3700  
Chicago, Illinois 60661  
312-606-3214  
mkling@hmblaw.com